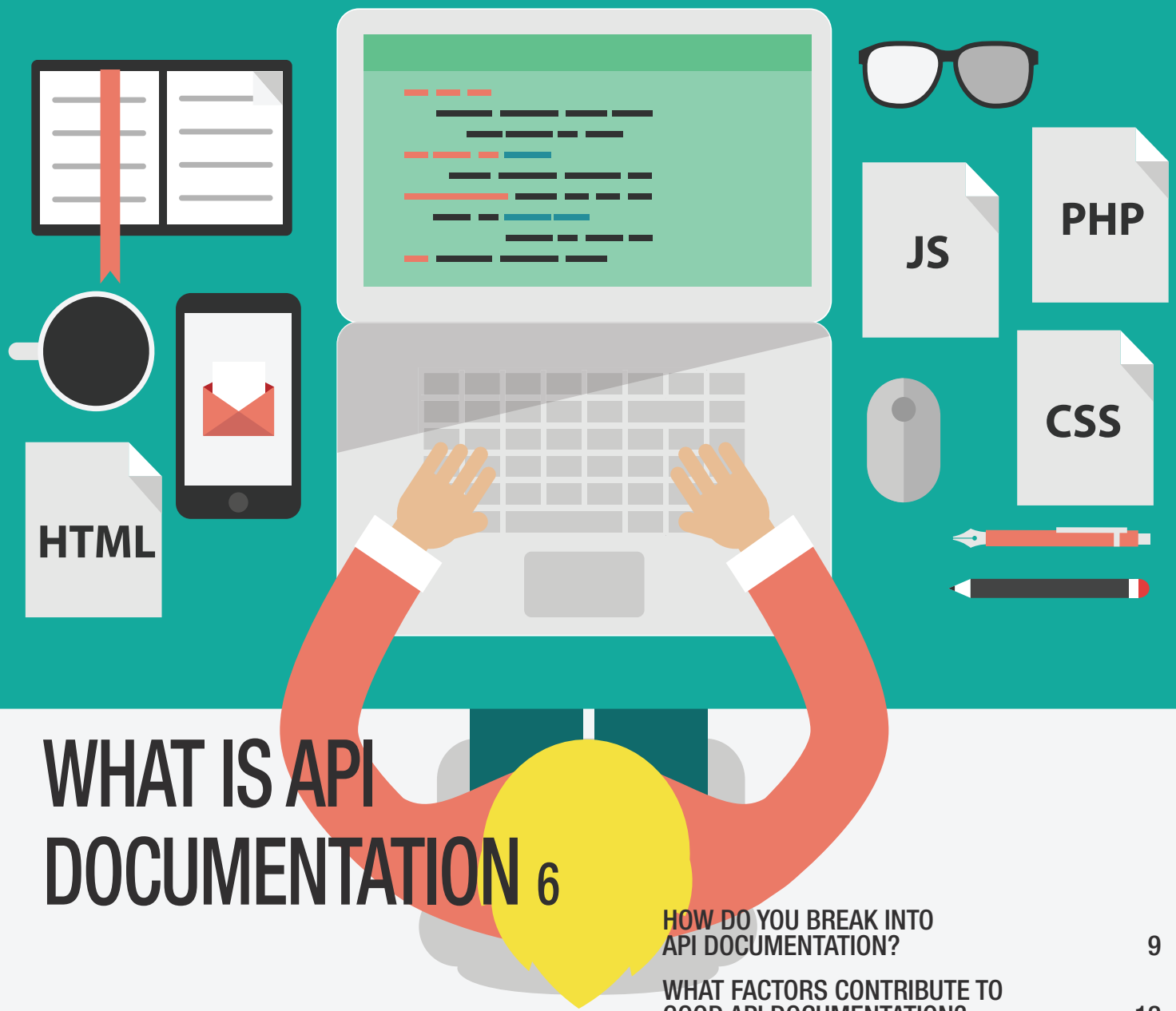
SEPTEMBER 2014

intercom

THE MAGAZINE OF THE SOCIETY FOR TECHNICAL COMMUNICATION



WHAT IS API DOCUMENTATION 6

HOW DO YOU BREAK INTO API DOCUMENTATION? 9

WHAT FACTORS CONTRIBUTE TO GOOD API DOCUMENTATION? 12

LESSONS LEARNED AS A NOVICE API WRITER 16

HOW TO WRITE HELPFUL CODE SAMPLES 19

HOW MUCH PROGRAMMING DO YOU NEED TO KNOW TO WRITE API DOCUMENTATION? 23

NEW ADVANCED WORKSHOP AVAILABLE

MADWORLD

The technical communication and content strategy event you don't want to miss.

SAN DIEGO | APRIL 12-14, 2015

FULL SCHEDULE NOW AVAILABLE

"This conference rocks! Can't wait to enhance my projects with all the new tips and trick I've learned."

—Chad Krieger | ***Policy and Procedure Analyst, Umpqua Bank***

FEATURED SPEAKERS

Matthew Ellison
UA, Europe

Derek Warren
Venafi, Inc.

Tony Self
HyperWrite

Angela Richer
Autotask Corporation

Laurie Pulido
eLearning Innovation LLC

Scott DeLoach
ClickStart

Allison Ellington
Travelport

Neil Perlin
Hyper/Word Services

OVER 40 SESSIONS AND MORE THAN 20 EXPERT SPEAKERS FROM AROUND THE GLOBE

Learn from Illumina, Blackbaud, Venafi, Autotask, Sabre, Travelport and more.

VIEW SCHEDULE VIEW SPEAKERS

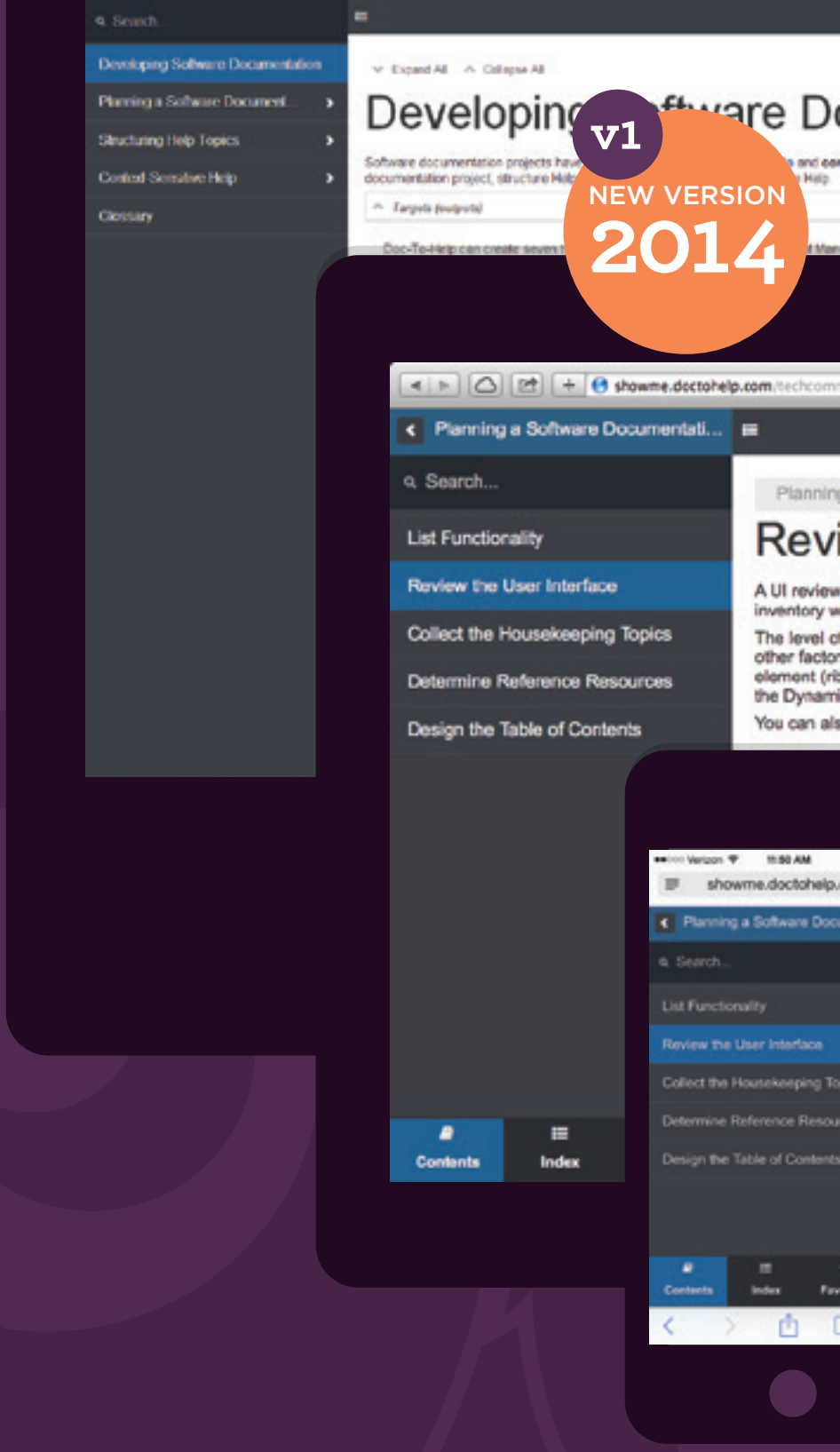
madcap software

One Output for Every Screen

Trusted by technical communicators all over the world to deliver professional-quality documentation, Doc-To-Help publishes technical instructions, policies, procedures, and training manuals for the web.

Gone are the days when web publishing only involved the traditional monitor. Now screen sizes range from three to 27+ inches and interactivity has moved from the traditional keyboard and mouse to touch.

Doc-To-Help has the solution for every situation: Responsive Help. This new output responds to screen size and adjusts itself accordingly. With Doc-To-Help's Responsive Help you need only produce one output for the web and it will work on any device.



doc·to·help®

Download a Free Trial @
doctohelp.com

©2014 GrapeCity, inc. All rights reserved. All products and brand names are trademarks and/or registered marks of their respective holders

September 2014

Volume 61 | Issue 8

4 **From the Editor**

4 **From the Guest Editor**

FEATURES

API DOCUMENTATION

6 **What Is API Documentation?**
By Scot Marvin
This article describes what an API is, what API documentation contains, the audience for this type of documentation, and how you can get started in this field.

9 **How Do You Break into API Documentation?**
By Mary Connor
An overview of the market for API documentation, including a description of the salaries, the working environment, the expected skillsets, and how to transition and train for an API career.

12 **What Factors Contribute to Good API Documentation?**
By Lois Patterson
The author outlines some considerations for evaluating whether your API documentation efforts result in good documentation that is usable and helpful to the audience.

16 **Lessons Learned as a Novice API Writer**
By Mary Linderman
The author shares some of the lessons she learned working as an API writer, including terminology, working the development team, using the technology, learning about class libraries, and understanding stakeholders and users.

19 **How to Write Helpful Code Samples**
By Sarah Maddox
How can technical communicators write code samples? This article describes the nature and purpose of code samples, and gives guidelines on how to write them.

Continue Your Learning with STC Education!

STC offers a wide variety of online education options throughout the year. Whether you need an introduction to a subject, an in-depth review, or just a brush-up, STC has what you need. Advance your career with STC's varied collection of online education.

Live Web seminars

Multi-week online courses

Virtual conferences

Recorded webinars

And *free* archived seminars (members only)

23

27

29

23 How Much Programming Do You Need to Know to Write API Documentation?
By Samuel Wright
One of the biggest challenges technical writers have in writing documentation is learning to read code. How much programming is needed? The author explains how to pick a programming language and a project, how to write good code samples, and provides further resources.

Note: Additional articles on this topic may be found online at *http://intercom.stc.org*.

SOCIETY PAGES

25 International Summit Awards Best of Show and Distinguished Winners Announced
By Nathaniel Lim

26 2015 Membership Season is Open!

26 Reminder of Deadlines for Awards and Honors

27 STC Summit 2015: Call for Proposals Open

COLUMNS

ETHICS
29 Holiday Madness: Tough Totes Shoulders (Un)Successful Startup
By Heidi L. Everett

STUDENT PERSPECTIVES
32 Communities of Practice for the Workplace Environment
By Alana Baker

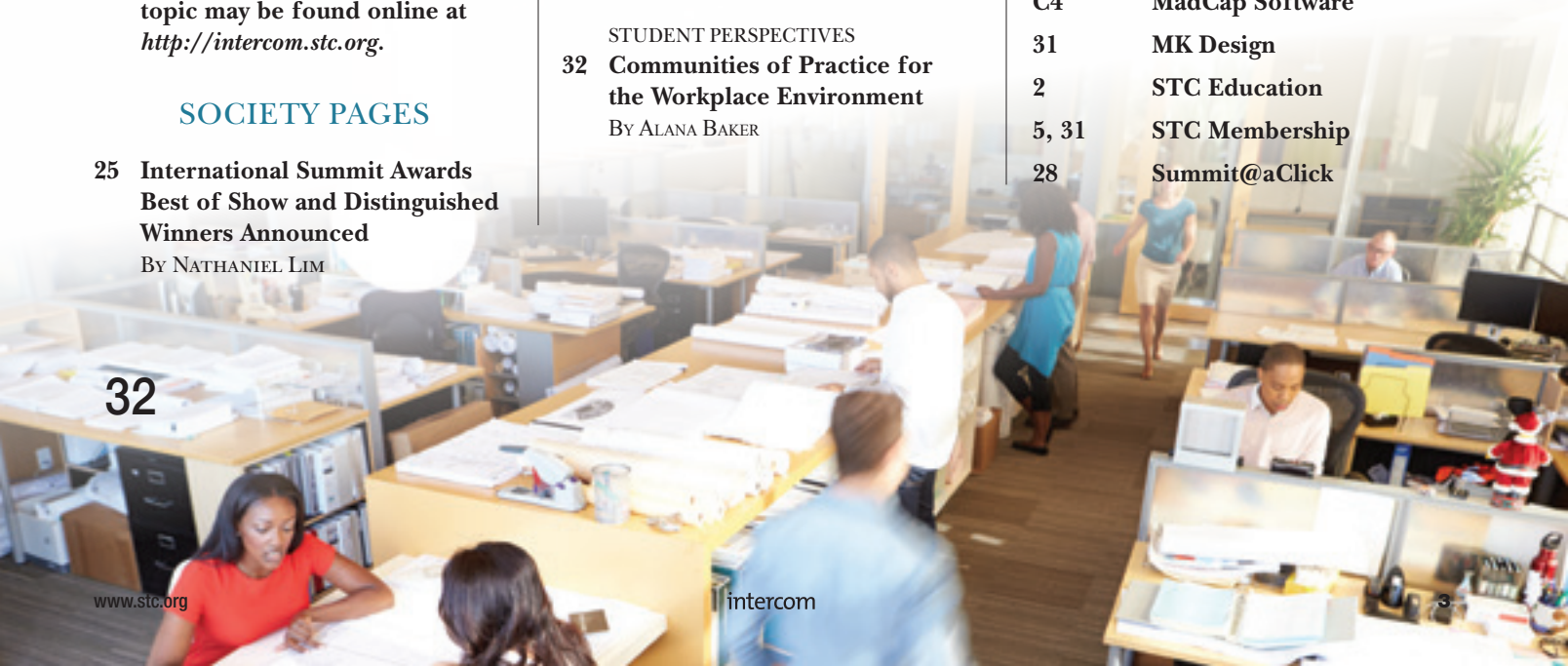
DEPARTMENTS

FYI
35 Mark Your Calendar
Organization Events Across the Globe

IN MEMORIAM
36 Bill Owen Coggin (1948-2014)
By Bill Leavitt

ADVERTISERS

C3	Adobe
C2	Doc-To-Help
C4	MadCap Software
31	MK Design
2	STC Education
5, 31	STC Membership
28	Summit@aClick


32

intercom

Intercom, the magazine of the Society for Technical Communication, is published to provide practical examples and applications of technical communication that will promote its readers' professional development.

Publisher Society for Technical Communication
Editor Elizabeth E. (Liz) Pohland
Assistant Editor Marisa Seitz

CONTRIBUTING EDITORS
Scott Abel, *Meet the Change Agents*
Andrea Ames and Alyson Riley, *The Strategic IA*
Thomas Barker, *The Academic Conversation*
Nicky Bleiel, *The Essentials*
Jack Molisani, *Advancing Your Career*
Neil E. Perlin, *Beyond the Bleeding Edge*
Linda Roberts, *All Access*
Derek Ross, *Ethics*
Kirk St.Amant, *Student Perspectives*
Brian Still, *Trends in Usability*

EDITORIAL ADVISORY PANEL
Beth Agnew, Barrie Byron, Liz Herman (Chair), Alan Houser, Paul Mueller

STC BOARD OF DIRECTORS
Officers
Kit Brown-Hoekstra, *President*
Bernard Aschwanden, *Vice President*
Nicoletta A. Bleiel, *Immediate Past President*
Jane Wilson, *Treasurer*
Alyssa Fox, *Secretary*
Directors Charles Fisher, Ray Gallon, Deanne Levander, Ben Woelk

SOCIETY FOR TECHNICAL COMMUNICATION
9401 Lee Highway, Suite 300
Fairfax, VA 22031-1803
+1(703) 522-4114 | +1(703) 522-2075 fax
Send questions and correspondence to the editor at the above address. General information about STC: stc@stc.org *or* www.stc.org.

Address Changes: membership@stc.org
Intercom Editorial: intercom@stc.org

Reprint Permissions: www.copyright.com
Intercom (ISSN 0164-6206) is published 10 times a year (monthly, except for August and December) by the Society for Technical Communication, 9401 Lee Highway, Suite 300, Fairfax, VA 22031-1803, a nonprofit educational organization, as a service to its membership. Membership in STC includes a subscription to **Intercom**. Periodicals postage paid at Fairfax, VA, and additional offices.
Postmaster Send address changes to **Intercom**, Attn: Membership, c/o Society for Technical Communication, 9401 Lee Highway, Suite 300, Fairfax, VA 22031-1803.

Copyright © 2014 by the Society for Technical Communication. All rights reserved. Printed in the USA.

All articles in **Intercom** are copyrighted by the authors unless otherwise indicated. Anyone interested in reprints, reproductions, or reuse must contact the authors for permission.
To advertise in **Intercom**, contact Stacey O'Donnell at *stacey.odonnell@stc.org*, Society for Technical Communication, 9401 Lee Highway, Suite 300, Fairfax, VA 22031.
Ph: +1(703) 522-4114; Fax: +1(703) 522-2075.

A Note from the Editor

I AM PLEASED to present a special guest-edited issue of *Intercom* this month on API documentation. Tom Johnson is a senior technical writer for 41st Parameter, a company specializing in fraud detection and device identity recognition. He writes about API documentation and other technical writing topics on his blog, *idratherbewriting.com*. He lives in San Jose, CA, and is a member of the Silicon Valley STC Chapter. Thank you, Tom, and all of the authors of articles in this issue, for such an interesting and valuable contribution to the field of technical communication!

—Liz Pohland
liz.pohland@stc.org

From the Guest Editor

FOR THE FIRST eight years of my technical writing career, I wrote GUI-based end-user documentation. By GUI-based documentation, I mean the kind of documentation that includes a user interface where users can interact by clicking buttons, selecting options from lists, browsing from tab to tab, and so on.

Writing GUI-based documentation was all right. I felt I could figure out any application in a short amount of time. By mere trial and error, I could learn the logic and workflow and write the help topics.

One day, my organization laid off all the technical writers. The market in Utah (where I was living) wasn't so hot, so I decided to move to the mecca of tech: Silicon Valley, California.

Silicon Valley, it turns out, has a lot of jobs for API technical writers. I'd been wanting to define a trajectory for my career that didn't lead to management, and since I felt confident in my technical abilities, I decided to take a job doing API documentation at a startup company specializing in gamification.

I soon realized why I'd been hired. The existing writer felt overwhelmed by a REST API and JavaScript SDK that she had to document. The company had been looking for a writer with strong development skills who could both write fluently as well as understand the depths of JavaScript code.

I studied up on JavaScript programming as best I could and managed to muddle my way through the writing tasks. As I worked on the documentation, particularly for the JavaScript SDK, I realized that I was entering a territory that was unfamiliar to me. My previous eight years of technical writing experience, including the numerous conferences I'd attended, really didn't prepare me for all the questions I had.

What tools should I use to document this API and SDK? How do I understand all of this programming code? How do I know if this API documentation contains useful information for developers? Will my audience know what this code does by merely looking at it, or do I need to explain it in detail? Do I put my documentation in the source code or another place?

I realized that API documentation, or rather developer documentation, is a sub-specialization in the tech comm profession—a specialization that doesn't have many resources or information to guide technical writers like me. In fact, about the only book on API documentation is an out-of-print book written nearly 8 years ago (which dives into Java instead of REST APIs, which are now more common).

The most relevant conference for those doing API documentation, *WriteTheDocs.org*, is so removed from regular tech comm events that the participants (many of whom are developers) refer to technical writers as "documentarians."

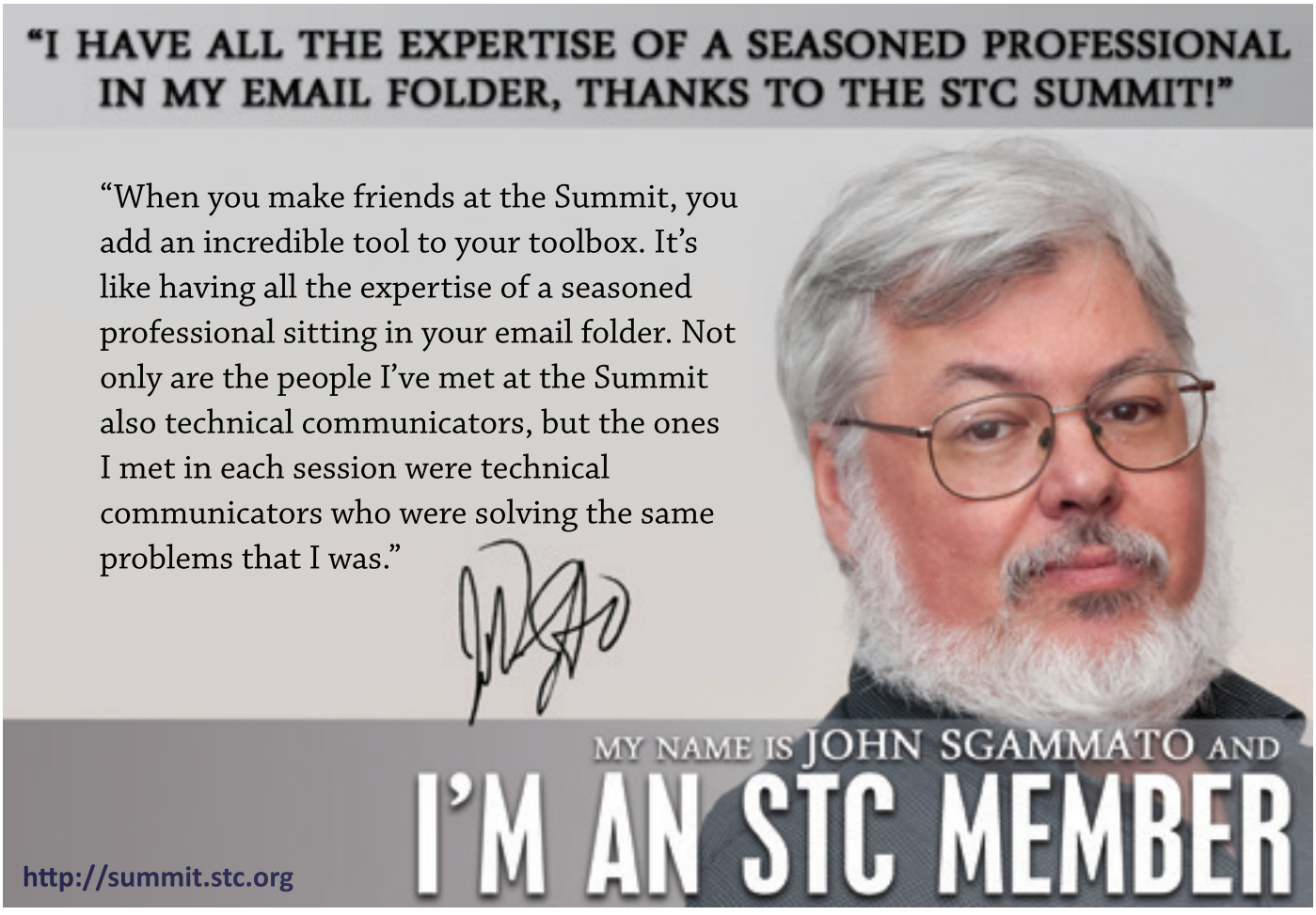
APIs are proliferating on the Web more than ever before. *Programmableweb.com*, a site that tracks and covers Web APIs, has a directory of more than 11,000 Web APIs. If you look at the documentation for these APIs, very few of them are the same in terms of format and style. Their approaches differ as much as any Wild West frontier you can imagine.

There is a great need for technical writers who can speak code fluently enough to document all the APIs coming out of Silicon Valley and elsewhere. While Java has traditionally used Javadocs for source-generated documentation, and C++ has used Doxygen, and .NET has used Sandcastle, REST APIs are a breed of their own, with no tool facilitating source-generated documentation. They present a lot of opportunity not only for communication expertise, but also creativity in layout, display, and organization.

This issue of *Intercom* is full of articles about API documentation. Beyond articles included in the print edition, you'll also find additional articles about API documentation in the online version on the *Intercom* website (*http://intercom.stc.org*). I hope these articles help increase much-needed instruction about API documentation in our field.

If you've been thinking about taking your career in a new direction, consider specializing in API documentation. Not only will it challenge and expand your technical abilities, it will also give you a new landscape to play in, job security, higher salary potential, and a host of challenges to solve.

—Tom Johnson
tom@idratherbewriting.com


"I HAVE ALL THE EXPERTISE OF A SEASONED PROFESSIONAL IN MY EMAIL FOLDER, THANKS TO THE STC SUMMIT!"
"When you make friends at the Summit, you add an incredible tool to your toolbox. It's like having all the expertise of a seasoned professional sitting in your email folder. Not only are the people I've met at the Summit also technical communicators, but the ones I met in each session were technical communicators who were solving the same problems that I was."
MY NAME IS JOHN SGAMMATO AND
I'M AN STC MEMBER
http://summit.stc.org

shutterstock.com/Oberon_sk

What Is API Documentation?

By SCOT MARVIN | *Member*

Sometimes software developers want their software to be used by another software application, instead of being used by a human being. … [A]n API makes it possible for software applications to communicate with each other.

I'VE BEEN WRITING API documentation for 15 years. During that time, I've seen a rise in API offerings. Companies are producing platforms that they hope others will plug into, using the platform's API. API growth means more API documentation. And that's a good thing for technical communicators.

In this article, I'll address what an API is, what API documentation contains, the audience for this type of documentation, and how you can get started in this field.

What Is an API?

Sometimes software developers want their software to be used by another software application, instead of being used by a human being. Because the interaction will be with another software application, instead of with a person, there needs to be an agreed-upon way for the programs to interact. This language, and its inherent set of rules, is called an application programming interface (API). Essentially, an API makes it possible for software applications to communicate with each other.

The information exchange between your software and an API usually takes the form of a request and a response. Your software makes a request, using the API, and the other program sends a response back using the API.

Some APIs are used to exchange data between products that are external to one another. If you want to interact with Google or Twitter, you would use their API documentation as a reference for writing code to exchange data between your product and their products.

Other APIs are used to exchange data internally between services within a company or product. Both Windows and Android operating systems have internal APIs so that developers can easily write sub-programs that communicate with different parts of the system.

And still other APIs are used both internally and externally. Amazon Web Services has over 40 services that offer APIs for anyone to use. Amazon developers also use these APIs internally, so that the various services can communicate with one another.

Whether used externally or internally, APIs need to be documented so that communication can take place.

What Is API Documentation?

APIs are diverse in terms of language and functions, but what they share is the fact that the user sends a request and the API (sometimes) returns a response. The technical writer's job is to detail what goes into a request, what form the request takes, and what data gets returned.

The structure of API documentation is varied but, in my opinion, a good API narrative provides readers with:

- syntax of each operation
- a description of what the operation does
- what parameters the operation takes, including default values, valid values, and type of data—boolean, string, etc.
- what data the operation returns
- error messages you might encounter using the operation
- examples of working requests and responses

This sounds like a lot, but all good code has the content already there. The API writer has to look, ask, and act like an editor to deduce content that is not in the code.

At its core, API documentation details information about code: the available commands, what each command does, and the parameters for each command. Traditionally, an API document describes one operation and how to make a request. This document is usually composed of tables filled with parameter names and accompanying values, along with sample code. Combine all the API documents, one for each function in an API, and you have produced an API reference guide. It's basic but helpful.

But what about documentation for people who want to do more than just make a simple request? Most developers want to know how to string commands together and interact with input and output. Many writers produce developer guides that contain common use cases and tasks for working with an API, along with code examples.

Another type of API document, the user guide, contains content that's not related to coding, but rather explains how to use an API's associated command line interface or graphical user interface. These guides normally contain concepts needed to understand the API, deployment and configuration tasks, and best practices.

If a writer wants to lower the barriers for using the API—essentially helping anyone who owns a computer to use the API—he or she will create a "getting started guide." This type of API document helps users understand the API by walking them through a set of tasks, after which they know more about the API operations and what they could do.

Some companies package these various API-related documents together and publish them as a software development kit (SDK). People using an SDK usually count on having other things included besides just API documentation. Normally an SDK includes working code and a test platform of some kind, along with the documentation.

Ultimately, readers of API documentation are looking for two things: correct information and working sample code.

Who Uses API Documentation?

In technical communication, we usually place audiences into three broad categories: end users, system administrators, and developers. I've come to believe that every reader of a technical document, whatever the document is, is really just an end user of some type. Some end users need information about how to create a new file in an application. Other end users need information about how to run a command in a terminal or command prompt. And other end users need information about how to write a program that exchanges data with another program.

End users of API documentation are usually developers, but there is API documentation for system administrators, too. System administrators usually look to API documentation to perform tasks that don't need to be programmed but need to be used on an as-needed basis. Traditionally this has been with command line operations. For example, to set up a user in an account, the system administrator would run a command to generate a key pair so that the user could start making requests to the API.

A developer, on the other hand, might want to create a string of operations that run programmatically at certain times or in certain conditions. For example, when a request from one command is returned, a second command will then run using the returned values from that first command. In this case, a developer builds a workflow or a framework that uses the API functions.

Some APIs are meant for occasional commands, such as "backup my database at this point in time." Others are meant to be used in a repeated and programmatic way, such as "when my Web server gets 80% busy, create another Web server and share the workload between the two servers."

Who Writes API Documentation?

Folks in our growing industry who write API documentation often have titles like "programmer writer" or "senior technical writer." Usually there are two paths to becoming a writer who documents APIs. The first path is the one taken by the writer who has written for a long time, has a great understanding of communication practices, and decides to start documenting APIs. This is the learn-code-as-you-go path.

The second path is the one taken by the developer who has decided to concentrate on writing. This is the learn-writing-as-you-go path.

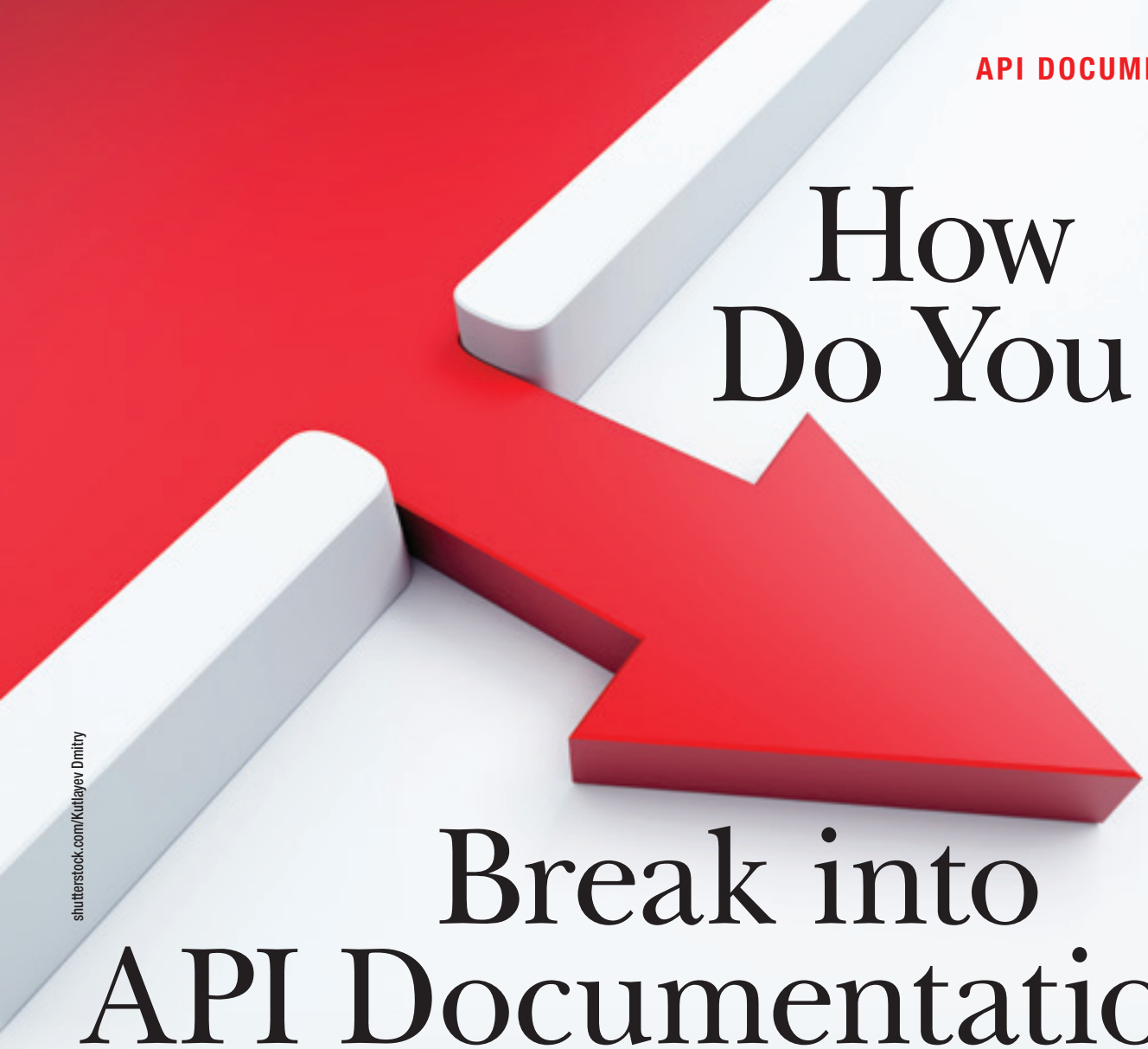
This article is for the first path sojourners. If you're interested in documenting APIs, you would be best served by the following general rules:

- **Learn to read code.** And by that, I mean learn to read code, not necessarily write it. Look around and read the comments in the code. See if you can make sense of what the code does. Writing API documentation requires some familiarity with a programming language, but you don't have to be a developer. You have to understand what to look for in code, sure. But mainly you need to understand what the code does, why it was written, what the intended purpose is, and who should use it and in what situations. This last part is the most important, because that's the part that readers will keep going back to your documentation for. Consider getting an introductory book about the programming language for the API you're interested in.
- **Learn common API technologies.** Normally, the API message content is either in Extensible Markup Language (XML) or JavaScript Object Notation (JSON). The format of either file type is pretty easy to comprehend, but it's good to look into an introductory book about these. The API content is delivered using Representational State Transfer (REST), or possibly Simple Object Access Protocol (SOAP). You can learn about these quickly using a good book or a search on Google.
- **Look at a few different types of API documentation.** It's always good to see what is available. Check out the documentation for Amazon Elastic Compute Cloud documentation (*https://aws.amazon.com/documentation/ec2/*), Google Cloud Storage XML API (*https://developers.google.com/storage/docs/xml-api-overview*), and Twitter (*https://dev.twitter.com/docs/api*).
- **Learn to swallow your pride.** Ask a subject matter expert what's going on in the code. There are occasions when even the best developers have to consult one another to figure what some code does. If it's not always clear to them, it won't always be clear to a writer. But remember to be prepared first. Don't waste developer time because you didn't do your research first.
- **Grow your curiosity.** Play around with the code with a toolkit or a test harness, if such a thing is offered for the API. Send a request using the API and see what happens.

It sounds like a lot of work and a lot of things to learn, but it really is pretty simple once you get started. API documentation is a growing area in our craft. I encourage you to check it out and see if you like it. We need a lot more great writers.

Ultimately, readers of API documentation are looking for two things: correct information and working sample code.

SCOT MARVIN *(*scotmarvin@gmail.com*) writes API, cloud administration, and system administration documentation for Eucalyptus Systems, a company that makes private and hybrid cloud software that is compatible with Amazon Web Services. Scot lives in Corvallis, OR. When he's not chasing his toddler and teenager around, you can find him on Twitter (*@scotmarvin*) or on the Web (*scotmarvin.com*).*



shutterstock.com/Kutlayev Dmitry

How Do You Break into API Documentation?

By MARY CONNOR | *Member*

The Market

Large numbers of us broke into technical writing by documenting software that runs on personal computers. Over the years, we have watched sales of those machines and their software packages stagnate. The manuals and help that we created grew gaunt under the relentless pressure to do it faster, cheaper, and lighter. With the rise of support-written knowledge bases (think ZenDesk, *www.zendesk.com/*) and community-generated content, many wondered if conventional documentation jobs were destined to fade.

Now we're witnessing dizzying growth in high tech, driven by the shift to ubiquitous computing through mobile devices and by distributed, cloud-based storage and services. All of this growth depends on there being solid and successful APIs to interconnect our devices and services. A quick look at these resources shows that the range of API offerings is stunning:

- API Directory (ProgrammableWeb, *www.programmableweb.com/apis/directory*)
- Government APIs (data.gov, *http://catalog.data.gov/dataset?res_format=api&_res_format_limit=0*)
- Open APIs (Wikipedia, *http://en.wikipedia.org/wiki/List_of_open_APIs*)

Just how successful these APIs are has everything to do with how quickly and easily programmers can become power users of these new interfaces, and so everything to do with documentation and tutorials, and us! That's the good news: the job market for programmer writers has become habañero hot. Since I've put full-time API documentation experience on my profile, I've started receiving an eye-popping stream of unsolicited invitations for positions with tech companies that are everyday names to the wired crowd.

The Salaries

Not only are *more* API jobs becoming available, but these are jobs that pay *significantly* higher than conventional ones. In my experience, you would command hourly rates that are 1.5 to 2 times that of non-API jobs. And, while not a

salary issue, another feature of these jobs has tremendous monetary value: API jobs are more likely to offer the option of working remotely. However, the on-site jobs do tend to be concentrated in specific metro areas (such as San Jose), which have a higher than average cost of living.

Perhaps more relevant to our Great-Recession-sobered perspective, being an API writer means being a sought-after talent that's in relatively high demand, with broad options and mobility. And acquiring the skill-base of programming writing is *itself* insurance, as it grants access to jobs in more technically demanding shops, as well as job titles with more technical requirements, such as business analyst. Any way you look at it, your salary security improves.

The Working Environment

I associate the working environment for API writing with being embedded with the development team that codes the API. If you have worked within feature-based teams or on an Agile/SCRUM team, you are already familiar with this dynamic, being the "lone writer" that supports a group of developers and testers, usually in service of a product owner or product author. Your supervisor, however, may be the development manager. Having a developer for a boss can have huge implications for the expectations you face, the tools you get, the deliverables you own, and the content strategy that might be missing or discounted.

The most common Myers-Briggs personality type I've encountered on these teams is INTP, a fact that I bring up for this reason: your ability to relate to these developers and to win their respect means your success or failure in this job. All things are possible, but I think highly extroverted writers, for example, could face an uphill battle with team acceptance. I have witnessed several writers be rejected by developers who were repelled by their manner, and it's not a situation that just resolves itself with time and friendly words. If you don't seek nerdvana among deep thinkers who will call you out on your logical errors, you might want to reconsider this path.

The Skillsets

As an API writer, you can find yourself exercising a surprisingly broad set of skills, a set that you will find either exhilarating or terrifying (or both, if you're human):

- **Development—**You will need to be able to use the shop's chosen IDE (such as Visual Studio or Eclipse), work with their code files, run local builds, master their version control system, learn their code commenting style, and install, update, and *own* the API reference generation. Yes, unfair as it sounds, you will be expected to tackle all of the programmer and QA tools in addition to your own documentation tools.
- **Business Analysis—**You will find yourself in the position to serve as a business analyst for the interface. In practical terms, this means using your communications expertise to devise and argue for stronger, intuitive, and consistent object naming and organization, and using your research to compare the API both with competitors and with implicit or explicit industry standards.
- **User Experience—**As with all documentation jobs, you will still strap on your gloves to fight for a better user experience. In the case of developer documentation, you will look for ways to improve TTFHW (Time to First "Hello World"), how API users get introduced to the product, how they get set up, how they learn, and how they get help, both from support and their peer community. Chances are, you will be the one person on the team with the drive and experience to make headway on UX/CX, perhaps getting involved with support initiatives.
- **Technical Communication—**As if this all weren't enough, you still need to be an excellent communicator, such as Andrew Davis describes so well in his article, "How to Find Good Programmer Writers" (*www.contentrules.com/blog/how-to-find-good-programmer-writers*). Beyond your powerful writing, bring the full package of skills that separates the great from the competent: be empathetic, responsible, curious, humble, resourceful, attentive, autodidactic, passionate, precise—the writer that every project manager would weep to lose.

The Transition

To transition from a regular technical writer who documents GUI apps for end users to a technical writer who creates API help for developers, you need to be well along the path to being a developer in your own right. Yes, a developer, even if you've never taken a programming course in your life.

Happily, most documentation jobs offer several key areas in which you can develop your technical chops while serving the larger good:

- **Authoring Tool Plumbing—**Be the person on the team who knows the authoring tool inside out and backward, who can install it, upgrade it, back up and restore it, configure it, and import and export its content. Be the Web researcher who finds out how to leverage its obscure features to solve a problem in your help system; be the designer who braves editing the complex template files to add Google analytics or to achieve special Javascript effects that make the CEO take notice. Be the tool agnostic who can do proof-of-concept projects in any number of new and unknown tools, without any handholding and on your own initiative. Be the analyst who studies other help systems for useful enhancements and persuades a developer to write you a script that will help you implement one on your own.
- **Build Engineering—**Be the person on the team who figures out how to automate your help builds, even if the tool doesn't support it directly. Find ways to integrate the help source, builds, and outputs into the existing build process used for the code. Figure out how to publish draft help internally as well as in beta and general availability(GA) versions. Come up with a way to fake conditional build behavior so that internal staff have access to additional content that will never be released.

Be the person who never stops finding new opportunities to single-source content across file formats and disparate uses. Be the one who learns enough about FTP to automate the publication of content to the Web.

- **Batch and Automation**—Be the person who notices that manual doc processes are being repeated and find ways to automate them, starting with simple DOS batch files and Windows Task Scheduler. Improve upon processes that require you to remember to do the right thing in the right sequence, such as to copy files to network locations or manually rename files to show status; find a better way. Use the scripting tools built into your business apps (such as Visual Basic for Applications) to automate steps and transformations that improve documentation quality, such as Word macros to correct all sizing and formatting problems in hundreds of screenshots. Make full use of variables and templates in your tools, and explore all of the fields available, along with the parameter switch options they offer.
- **Internal Developer Content**—Be the person who volunteers to help write and edit wiki pages to guide new developer hires. Attend every language/tool/methodology training that your shop offers to the development group. Offer to manage, edit, organize, and publish the development artifacts for the group. Help business analysts create clearer and lovelier diagrams and slide decks, and study the content as you go. Attend "brain-dumps" as a scribe, and post your notes and Web conference recordings in the repository (creating it, if need be). Be the go-to person for knowledge management strategy.

What I described above is the path that I journeyed down, which led me to take over API reference projects at my job and then to transition to full-time API documentation responsibility. If those descriptions leave you cold, sweating, and feeling as if you'd want any job but that, you might not enjoy being in the hot seat as an API writer.

The Training

If you are interested in the API career path, strike the word "training" from your consciousness and substitute the word "studying," because the culture of code development is one of unceasing self-education. Even when developers nominally finish training courses on new languages and methodologies, they know that they are never done researching and improving their grasp of the material, which itself will never stop mutating into new versions and forms. Development's tribal code is all about building and maintaining your own technical credibility, without hand-holding.

To make this more real for you, I'll describe how I've witnessed API developer interns being treated. They are assigned to serve a specific team member, who piles them with tasks and problems to solve. To a fair degree, interns are allowed to twist in the wind, to struggle to get their environments working and their projects moving. What at first seems sadistic reveals itself to be highly illuminating: the team gets to observe how quickly the interns teach themselves the new domain, how resiliently they study, acquire help, and handle frustration, and how cleverly they conquer the challenges blocking their projects. Even if the interns resent the intellectual hazing, they emerge wiser and full of confidence that they can handle their next job assignments. They join the tribe.

So, to be a successful API writer, be prepared to join the tribe. Think of yourself as a developer newbie, not one that will threaten to take the job of anyone on the team, but one who will learn just fast enough to be able to teach other newbies. To start, take whatever programming courses are available to you; community colleges and school district community education programs are great resources. But don't let logistical challenges stop you—you can now learn programming completely on the Web, sitting on your couch. Try out courses from the sites below so that you can find some that fit your learning style (see this excellent survey: *http://designzum.com/2014/03/07/best-resources-to-learn-code).*

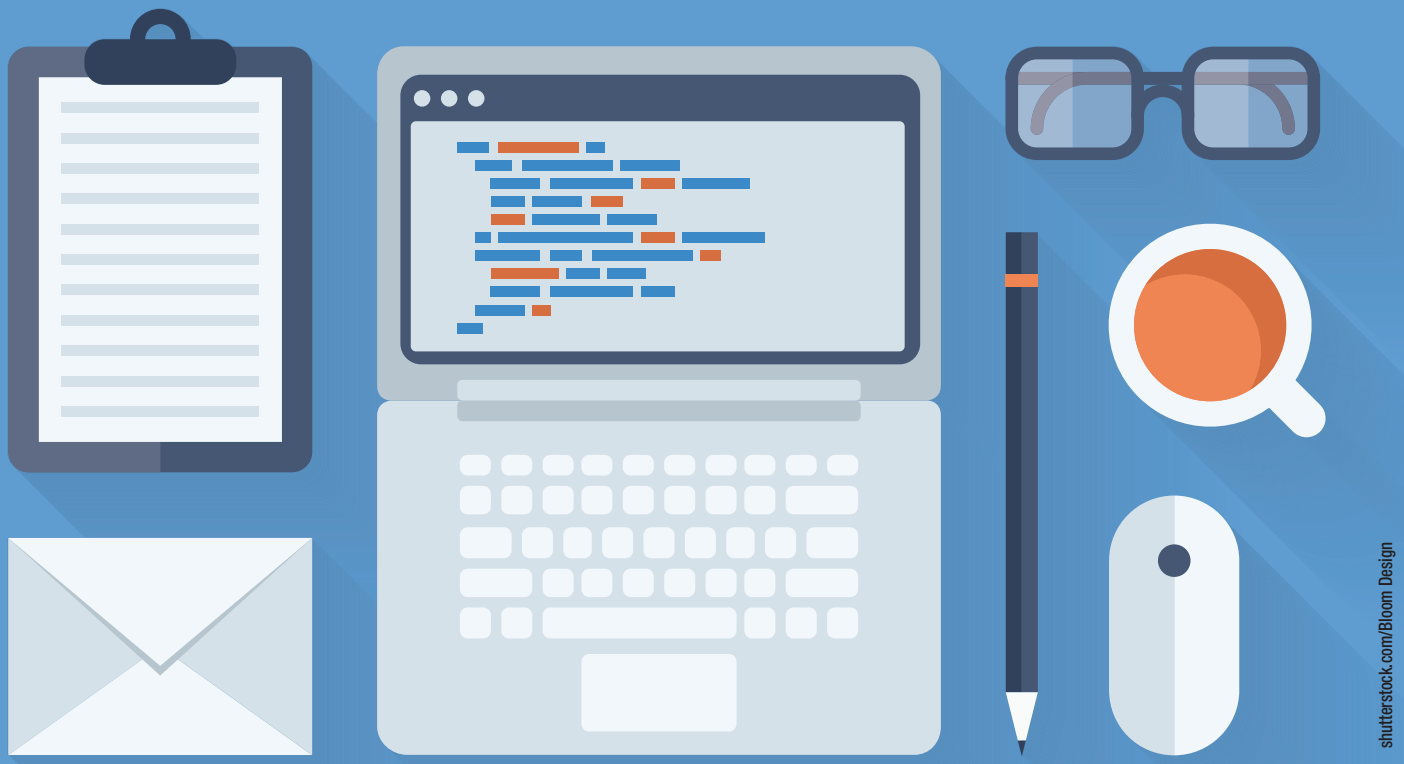
- Coursera (*https://www.coursera.org/*)
- OpenCourseWare Consortium (*http://ocwconsortium.org/*)
- Codecademy (*http://codecademy.com/*)
- Code Avengers (*http://codeavengers.com/*)
- Code School (*http://codeschool.com/*)
- Treehouse (*http://teamtreehouse.com/*)
- LearnStreet (*http://learnstreet.com/*)
- Udacity (*https://udacity.com/*)
- CodeHS (*http://codehs.com/*)
- Khan Academy (*http://khanacademy.org/cs*)
- Scratch 2.0 (*http://beta.scratch.mit.edu/*)
- SQLZOO (*http://sqlzoo.net/*)

As you complete coding courses, look for opportunities to code solutions to small problems at work, church, or home, and remember to record your completions and outputs on your resume and LinkedIn profile. At the same time, start following the blogs and sites of more technical tech writers, and join relevant LinkedIn groups, such as the one for API documentation (*http://linkedin.com/groups/API-Documentation-3709151*).

The Payoff

Is API documentation more rewarding and fulfilling than creating other types of documentation? For me, yes, for surprisingly non-career reasons. I love having dared to face my personal Everest, my fear that I wasn't capable of doing this work. I love being embedded with developers and testers as a their supporter and peer. And, sweetest for any tech writer, I love that these users, these API users, actually RTFM.

A past president of STC Austin, Mary Connor has an MA in English and has been performing and teaching technical communication for 20 years. Currently at 3M, she has created developer documentation and training materials for REST/SOAP services at Telogis and several generations of API products at Advanced Solutions. She is keen to capture tribal knowledge and to single-source/automate documentation production, which opens space for the important stuff: powerfully clear writing and diagramming. You can contact her through her blog at www.cleverhamster.com.



shutterstock.com/Bloom Design

What Factors Contribute to Good API Documentation?

By LOIS PATTERSON

YOU MAY BE FAMILIAR with what API documentation is and understand the organization of reference information into logical arrangements. But what makes API documentation good or bad? How can you, as a technical writer, evaluate such a developer-oriented product?

In this article, I'll outline some important points to consider for ensuring your API documentation efforts result in good documentation that is usable and helpful to the audience.

Start with Good, Consistent API Design

Perhaps, in principle, the design of an API is not the technical writer's task. In practice, if you are involved at the design and specification stage, you quickly get an idea about what the API is meant to do, and can spot pitfalls and inconsistencies early on, which is one of the primary ways that a technical writer adds value, in addition to simply producing the final API documentation.

How do you know if the API is well designed? A well-designed API straightforwardly manages common use cases and is as consistent as possible in its attributes, methods, and behavior. To develop use cases for validation purposes, think about the API as you would any software application. An API is essentially a software application that is used by other applications, rather than directly by the user. Think about the developer user who is creating the application that consumes your API.

For example, with the Netflix API documentation (*http://developer.netflix.com/docs/REST_API_Reference*), the documentation specifically considers use cases like "Searching for titles" and "Searching for people," reflecting that the API was originally designed with these specific use cases in mind.

You can develop the outline for your API documentation as soon as you have a minimal idea about what the API is meant to do. In fact, explicitly indicating the purpose of the API is of fundamental importance, and working with the API developers to polish this introductory and high-level material will help focus the development of the API.

Consistency is of primary importance to understand an API and to develop an intuition about it. For example, if your API allows you to create objects, such as boats and cars and trucks, with specific attributes, the API should reuse as much of the same terminology as possible between the objects, and the methods should be as similar as possible. If your API used the attribute "owner" for all three object types, that would be consistent. If, on the other hand, you used the attribute "vehicle_owner" for cars and trucks and "owner_boat" for boats, the developer user is likely to find your API non-intuitive.

Similarly, using a delete method for all three object types is consistent, but if you instead used "remove" for cars and boats, and "delete" for trucks, the developer user would likely become confused. And you, as the technical writer, would observe that what could be a nicely organized structure is, in fact, confusing.

Inconsistencies tend to creep in when you have multiple API developers who do not talk to each other. Part of your role, along with quality assurance and product management, is to look for inconsistencies and flag them as early as possible. With REST APIs (*www.restapitutorial.com*), a certain level of consistency is enforced—one reason they have become so popular.

In addition to consistent language, APIs should demonstrate consistent behavior as well. When an operation is performed, a message should always be returned, and when performing similar operations, the returned messages should be similar. The need for consistency is a well-understood development principle, and you should feel comfortable insisting on it.

Descriptive error messages can be very difficult to do, but consistent and detailed error messaging will help the developer user considerably. If the API responds with "invalid request," with no supporting details, the developer user will be stuck.

Include Code Samples to Address Variations in User Abilities

As with any other software documentation, audience analysis is key to determining what to say and how to present it in your API documentation.

Anecdotal experience, both from my workplaces and from conference sessions I've attended, suggests that, although you may have a proficient developer user in mind who needs only the barest information in order to successfully create an application that calls your API, the reality is that the developer user may be struggling with a language he or she does not know that well, may not have had past API experience, and in fact, may not be a developer at all. This makes the challenge of range even more difficult, as the optimal users must be treated well while still providing information that makes the proto-developers' lives easier.

At the previous Write the Docs 2014 conference in Portland, OR (a conference focused on API and other developer documentation), one presenter mentioned that he had found that marketers were adapting the JavaScript code samples to create their own apps rather than writing their own code.

We do similar copying all the time when sharing content such as videos. If you want to embed a YouTube video into a blog post, you just paste a link like this in your WordPress editor window.

```
<iframe width="560" height="315" src="//
www.youtube.com/embed/sumQULczJOg"
frameborder="0" allowfullscreen></iframe>
```

To obtain this code for any YouTube video, click the Share button, and then click the Embed link to see the code.

In the same way, many developer users (who consider themselves amateur Web page creators at best) copy code for Twitter streams, counters, and so forth into their websites.

When the process is this simple, it's inevitable that many non-developers will be using the API. And even if the process is not that simple, if your API has attractive features, inexperienced people will want to use it. Keep those inexperienced developer users in mind by creating friendly code examples they can leverage and customize in their own projects.

The snippets and the fully developed samples may be in separate documents. For example, the function documentation for the numpy (Numerical Python) function npv includes a full description and a simple example or two, but it does not demonstrate its use in an actual separate

application (see *http://docs.scipy.org/doc/numpy/reference/generated/numpy.npv.html#numpy.npv*).

Power users may simply consult the reference material, while more novice users may rely heavily on code samples.

Reduce the Time to "Hello, World!"

When you download a new application or program in a new language, there's a definite thrill and positive feedback loop if you can quickly perform a simple task, like the classic "Hello, World!" output example in programming tutorials. I suggest trying some popular APIs to see how quickly you can get to "Hello, World!," and take note of how the documentation helps or hinders you.

Making it easy to find the required documentation can be just as important as writing it well in the first place. I tried two popular open-source APIs—jQuery and numpy—and found that the useful and necessary information was not immediately evident.

jQuery

The jQuery site includes a basic example and explanation page here: *http://learn.jquery.com/about-jquery/how-jquery-works/*

The example almost works perfectly, but I had to figure out myself to rename the jquery<*version*>.js file in the downloaded files to jquery.js.

When I wanted to create an accordion widget in jQuery. I found a descriptive page with a good sample in the API documentation at *http://api.jqueryui.com/accordion/*.

This sample did not run at first because calls to remote JavaScript files cause security problems on some systems, so I changed the example to use local files. However, the API documentation did not explain this point.

I like that jQuery has assembled a comprehensive learning site. As the site states, "There's a lot more to learn about building web sites and applications with jQuery than can fit in API documentation" (*http://learn.jquery.com*).

numpy (Numerical Python)

The main numpy site tells where to download the latest version, a surprisingly important point:

- *www.numpy.org/*
- *http://sourceforge.net/projects/numpy*

Although I already knew that numpy is very much alive, seeing that the SourceForge download file was less than a day old inspired confidence. Having a product and documentation that looks as if they are maintained, with a recent update time, increases the user's confidence.

I ran into some installation issues and could not find an answer in the product documentation. An online search found a third-party site solved the issue: *https://datanitro.com/blog/2013/4/11/Python_on_windows/*.

Once the installation was complete, I looked through the function documentation, picked one to try, and copied the code sample into my Python editor. Here is a typical function documentation page, including an example: *http://docs.scipy.org/doc/numpy/reference/generated/numpy.sin.html*.

With the download, the Google search, and running the example, I got to "Hello, World!" in less than 10 minutes.

My experience highlighted the importance of recognizing that users may have forgotten some details of basic tasks. In this case, I had forgotten how to import modules from a package in a Python library.

Use the Right Platform and Tools

Because you rely heavily on API developers to help you document APIs, you should use tools and platforms that make it easy for them to contribute.

At Write the Docs 2014, one presenter spoke about how documentation flourished when he substituted simple tools for highly complex ones that discouraged teams from actually producing documentation.

Similarly, as described on the documentation page for the Cloudstack project, the documentation was originally written using DocBook XML, which produces beautiful documents, but is also complex and cumbersome. The project founder noted the lack of documentation contributors, compared to similarly situated projects, and decided he needed to use a simpler system. He substituted simple markup options like Markdown and ReST (Restructured Text) to create documents as marked-up text files, and used the Sphinx document generator to produce the output files (see *http://sebgoa.blogspot.ca/2014/03/migrating-from-publican-to-sphinx-and.html*).

If you are in charge of an API documentation project, position yourself as the manager of the project and a contributor, rather than as *the* writer since you will usually be using the work of many. Think about how you can foster a system where editing and contributing is frictionless.

Does this mean you should use wikis? Certainly Atlassian might argue for that, given that Atlassian uses its own Confluence wiki tool to document APIs (see *https://developer.atlassian.com/display/CONFDEV/Confluence+REST+API*).

At one time, some technology firms required all technical employees to use FrameMaker. Although I have not tried the latest versions, FrameMaker does not seem a natural fit for rapidly evolving Web-based documentation.

How about Google Docs? That does encourage collaboration, but the Google Docs format is not typically encouraged for publication. Version control can be challenging, too.

And the use of DITA, as useful as it may be in terms of facilitating content reuse and providing a framework for a navigable, searchable set of documents, will almost certainly discourage widespread collaboration among team members. In general, if you want API developers, professional services consultants, customer support analysts, and others to write useful content, they want a format that is easy to use and shows the results of their labors quickly to users.

So what options will work? With the rapid adoption of GitHub and Bitbucket for software projects (and not just

It is much easier to read code and understand what is going on than to write the code yourself (just as it is easier to read a great novel than to write one).

open-source projects), Markdown seems like a natural choice. Anyone who can edit content in JIRA or a wiki can write in Markdown, and Markdown is used to create attractive, functional documents for numerous software applications. Any contributor can use any editor to produce useful content. That said, Markdown, on its own, is not intrinsically designed to foster content reuse.

Doxygen and Javadoc are commonly used tools to create documentation from the structure of the code and from specially formatted comments in the code. Although these tools can often provide a useful summary of API documentation, they are rarely sufficient for producing a complete set of high-quality API documentation.

Swagger is a useful for browsing the functionality of an API. You can see an example of Swagger used to display a RESTful API here. Feel free to try it yourself at *http://petstore.swagger.wordnik.com/*. As with any other automatically generated documentation, a Swagger interface is typically not sufficient documentation, but it will provide a developer user a great deal of assistance.

Be willing to challenge your assumptions of what a good documentation tool looks like, and try experimenting.

Follow the Lead from Great API Documentation Sites

GitHub's API documentation is simple, but you can easily find everything you need at *https://developer.github.com/v3/*.

The Django REST Framework, which is used to create RESTful APIs, is not necessarily an easy application, but it provides a good example of a well-documented API (see *www.django-rest-framework.org/*).

Facebook's API is well considered (see *https://developers.facebook.com/docs/*).

Twitter has many useful features in its API documentation, including backlinks, which are an especially helpful navigation feature. For example, you can see all of the Twitter API documentation pages that link to the page that describes GET statuses/mentions_timeline at *https://dev.twitter.com/node/9526/backlinks*.

As mentioned previously, having examples of what can be done with the API should help your users considerably. For example, Flickr has an API Garden, which displays how developers used the Flickr API to create new applications at *www.flickr.com/services/*.

NPR (National Public Radio) displays some of the widgets developer users have created with its public API (see *www.npr.org/api/widgets.php*).

Even if these APIs differ radically from what your API, still take a look. Many concepts are common to all APIs. Looking at what is documented in other APIs will provide a sanity check to ensure you are covering relevant and significant topics.

Become More Familiar with Programming

Because APIs are typically used by developers, documenting APIs can seem forbidding, especially if you are not familiar with programming. One of the best ways to make yourself more comfortable with API documentation is to learn and practice programming, and to get used to reading code.

It is much easier to read code and understand what is going on than to write the code yourself (just as it is easier to read a great novel than to write one). If you are not familiar with HTML and CSS, those are markup languages that are easy to start with, and then you can move to any other languages you like, although JavaScript, Ruby, and Python are likely to be particularly useful.

Experimenting with an API is helpful to both the technical writer and developer users. Although RESTful (Representational State Transfer) APIs are not suitable for every purpose, they have become increasingly popular. A RESTful API uses several standard actions such as PUT, POST, GET, and DELETE that act on specific objects (also called objects or resources).

The cURL tool is suitable for experimenting with a variety of different API types (*http://curl.haxx.se/*). When this tool is installed, you can test API requests and responses, which is helpful for the technical writer, the quality assurance analyst, and the developer user who is trying to determine how to use the API's functionality in his or her application.

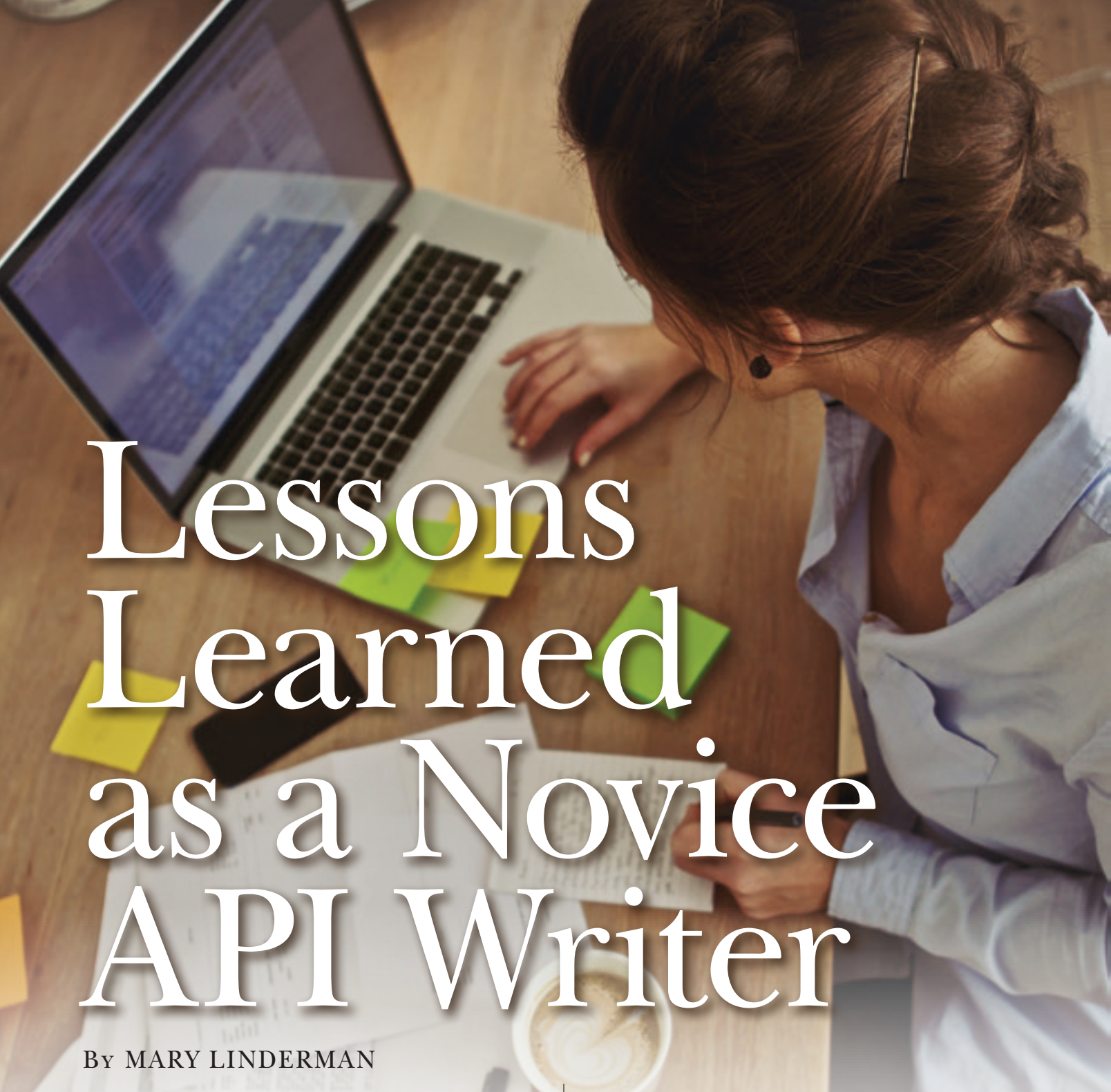
If you are a technical writer who understands software, you can understand APIs, too. The fact that APIs are becoming so prevalent is exciting, as software is becoming exponentially more powerful and capable, because any given application can use the features of so many other applications.

Conclusion

In conclusion, to make sure your API documentation is as helpful as possible for your audience, try to include the following elements:

- A clear statement of the API's purpose
- Automatically generated documentation for easy reference
- Good code snippets and good code samples
- Quickstart documents and tutorials (such as a "Hello, World!" tutorial)
- Searchable and navigable documentation
- Awareness and understanding of relevant programming concepts

LOIS PATTERSON *is technical publications manager at FINCAD and lives in Vancouver, BC, Canada. She has a strong interest in documenting technically complex systems. She enjoys the thrill of making software work. You can reach her on Twitter at @LoisRP or by email at* LoisRPatterson@gmail.com.



Lessons Learned as a Novice API Writer

By MARY LINDERMAN

IN THIS ARTICLE, I will share some of the lessons that I have learned working as an API writer for the past two years. I stumbled into this role by accident when my employer decided to encourage the customization of its software by offering a comprehensive API to third-party developers. While our documentation team had several good writers, I was the only one at the time who had taken any basic programming classes and I had a strong interest in the backend technology used in our software product.

Holding on to my limited knowledge of Java as a lifesaver, I plunged into the depths of API writing. My mission was to support our API team by writing comments for the class libraries, general descriptions of the API's underlying concepts, and tutorials illustrating how to use the API. I also worked on adding code samples provided by the development team to our website. With the help of another writer, we pulled off the impossible and posted our first version of the site in time for the release.

Since the first version of the site, we have revisited its content and presentation several times. Each version has involved long, hard hours of work during which I have reassessed my skills and abilities as a novice API writer. Here are some of the lessons that I learned as I have ventured out into the field of API writing.

shutterstock.com/Ammentorp Photography

Talk the Talk

When I started working on this project, the only documentation that existed was a few awkwardly constructed PDFs and some sparsely commented class libraries. My programming courses may not have made me a developer, but they had exposed me to the appropriate technical terminology that I would need to use when describing the programming constructs developed by my team. My familiarity with object-oriented terminology helped me gather information from my development teams since I could formulate focused and clear questions about classes, methods, properties, and other programming elements. I could talk the same talk as my developers, so I required less of their time for background information.

I also believed that the resulting documents had an authoritative and knowledgeable voice that communicated in language familiar to my developer community. Since the text served as the voice of the API, it needed to include the appropriate terms for the target audience. Familiarity with programming terminology also helped me to eliminate slang terms from the documentation, which might not be clear to third-party developers even though this secret language was second nature to my internal development team.

Embed Yourself in the Development Team

While developing our API documentation, my support system was a team of hardworking and technically savvy developers. Although I already had worked to develop rapport with my software engineers, the importance of having a strong and, sometimes, humorous relationship with the team was pivotal to documenting the API features that they built.

I made sure to attend team meetings, participate in team activities, ask questions, and actively listen to their implementation plans. Finding ways to complement or facilitate the development efforts of the team was a constant preoccupation for me during our release cycles. I reviewed specifications and researched technologies used by the team so that I could ask better questions and save the team time by focusing on their current development efforts. Since our organization uses an Agile programming methodology, I made sure to add documentation tasks for the team during sprint planning. I was like gum on the bottom of their shoes.

Whenever possible, I tried to underscore the importance of clear and accurate documentation as a way of showcasing their development work and increasing the accessibility of the API to third-party developers. I knew these efforts were really paying off when the team started coming to me with items that needed documenting and pointing out that we needed to add tasks for documentation during sprint planning.

Play with the Technology

One of the most difficult challenges that I have confronted was obtaining the resources and knowledge to play with the API. I have come to view documenting the API in the same light as writing about UI features. I needed to work with the API by running simple code samples to gain better insights about the user experience. As an actual consumer of the API, I felt that I could ask better questions about the code samples that were provided, and then write better descriptions of them.

Not everyone initially understood that providing the writer with the ability to consume the API would lead to better documentation. After all, the consumers of the API are developers, so the writer only needed to add the code samples to the documentation and provide descriptions. I sometimes found these assumptions slowly eroding the confidence that I had in my role, because I really didn't have the same level of technical expertise as our developer community.

However, I continued to develop my knowledge of the programming language used for our API, and the integrated development environment (IDE) used by our developers. I eventually obtained a development environment where I could use the API as well as help set it up so that I could run code samples. While I still spend more time reading the code samples sent to me than playing with them, I do have the tools that I need and I'm looking forward to writing some of my own code samples. Perseverance really does pay off.

Learn About Class Libraries

My experience with the API class libraries posed similar challenges to those that I faced when trying to consume the API. Developers use class libraries as a primary source of reference information about the classes, methods, and other features of an API. To create these libraries, developers and API writers add tagged comments to source code describing how to use the API, and then they pull these comments from the source by running automated tools over it.

I had used class libraries as a student in my Java classes, but I had never given much thought to the logistics of writing them, so I was only marginally familiar with the tools used for this purpose. Consequently, I was unable to provide much insight about how API writers collaborated with developers to add comments to the source, or about differences in tools available for this process. Since we were just beginning to create the class libraries, this knowledge would have been invaluable.

I recommend learning as much as you can about the content and organization of class libraries, as well as the tools used to generate them. I would suggest browsing through class libraries to observe the conventions that other writers use in their comments. Since some organizations only reluctantly allow API writers to comment on the source code, I have found that becoming familiar with the processes and tools used to create class libraries has fostered confidence in my abilities to contribute to this development effort.

Understand Your Stakeholder and Users

My role as an API writer had its genesis in the corporate goals set for the adoption of the API. I found that understanding these goals and their stakeholders has helped me adapt the documentation to meet the needs of my readers. Initially, the documentation was designed to educate external third-party developers about the API, but it also evolved into training material used for on-boarding new developers.

I have collaborated with product managers, instructional designers, and other stakeholders to learn about the community of users for this content. In addition, our documentation team has gained insights about the usability of our API documentation by gathering comments from on-boarding developers. We have asked them for feedback and invited them to participate in simple usability testing, so that we could gather insights about missing content, navigational issues, and others enhancements to improve the user experience.

Survey the API Scene

Everyone has an opinion about what makes a good API website. I have checked out websites for other APIs to compare different approaches to documenting this content. I have talked to our developers about API documentation that they found useful, including what they liked about it, and how they would improve it. I have gleaned useful information about different ways that developers consume our content from these discussions.

I have also found that not all developers learn the same way. Some like to start by reviewing the code samples, while others might look over conceptual information or diagrams. In the end, while I found some great ideas by reviewing different API sites, I concluded that the best approach to documenting the API is what made sense for my organization and readers. I also found that we have continually needed to evolve our site as the size and adoption of our API has grown, since we now have readers with even more varied skill sets and expectations about the information that we provide.

Trust Your Gut

One of the most important lessons that I have learned is to trust my own instincts as a technical writer. I currently rely on my development teams to provide code samples included in our developer's website. Occasionally, I come across a code sample or sample application that doesn't seem exactly right. It may not exemplify our best practices or it may be confusing to read. Clear code samples offer a great opportunity to demonstrate how third-party developers can effectively use the API.

I have reviewed code samples that would be helpful for an advanced user but may overwhelm a new developer just learning the API. Not all code samples are good teaching tools, so I have worked on providing my development teams with suggestions or standards for the code samples added to the documentation. My recommendation for these situations is to trust your gut by following up with the development team when you think a code sample doesn't work. While my technical persona has sometimes been intimidated by questioning the code samples, my writer persona has spurred me on in the search for clarity.

Cultivate Your Writing and Technical Skills

Since transitioning as an API writer, I have had the opportunity to interview other candidates for this position. Some candidates tend to emphasize their technical knowledge over their writing skills. The best candidates have paired strong writing skills with their technical knowledge, and they have demonstrated these abilities with samples in their writing portfolios.

In my role as an API writer, I have been required to write a wide range of content in addition to short comments found in class libraries. I have continued to cultivate my writing skills even as I work to expand my technical knowledge. This versatility has come in handy when I needed to create a high-level presentation about the API, standards for code comments, or use cases for specific features.

In addition, I have tried to stay current with new technologies that my organization may want to use in the future. I regret that I don't always have enough time to research and learn about new technologies, and I continually struggle with finding time for improving my skill sets.

When I first started working on the API, I would sometimes worry about how my technical background might hinder my ability to produce good documentation. I have learned that while I continually need to sharpen my technical skills, I shouldn't minimize the importance of the communication skills that I bring to this role. API writing offers a unique opportunity to balance technical knowledge with strong communication skills.

MARY LINDERMAN (linderman.mary@gmail.com) *is a technical writer at kCura Corporation in Chicago, IL. With over 10 years of technical writing experience, she explores new ways to communicate complex information to readers with a range of technical expertise. Mary's current interests focus on how corporations can leverage the strategic role that documentation plays in the adoption of APIs by third-party developers.*

REFERENCES

Gruenbaum, Peter. A Coder's Guide to Writing API Documentation, http://msdn.microsoft.com/en-us/magazine/gg309172.aspx.

Jacobson, Daniel, Greg Brail, and Dan Woods. APIs: Strategy Guide, O'Reilly Media, 2011.

Krug, Steve. *Rocket Surgery Made Easy*, New Riders, 2009.

Marshall, Ed. APIs and SDKs: Breaking into a Specialty Market, *http://tagungen.tekom.de/fileadmin/tx_doccon/slides/267_APIs_and_SDKs_Breaking_Into_and_Succeeding_in_a_Specialty_Market.pdf.*

How to Write Helpful Code Samples

shutterstock.com/Pressmaster

By SARAH MADDOX | *Member*

AS TECHNICAL WRITERS, we are trained to put ourselves into the position of the reader. In the case of API technical writing, the readers are developers looking to use our APIs and development tools.

To a developer, a code sample speaks a thousand words. Skimming code samples in a document is like skimming headings for the rest of us.

As technical writers, we are trained to create clear and meaningful content. And code *is* content. On the other hand, technical writers aren't necessarily software engineers. How can we write code samples?

We don't need to be code ninjas. The code in an illustrative sample is not the same thing as the production-ready code in an application. This article describes the nature and purpose of code samples, and gives guidelines on how to write them.

The code sample provides a stepping stone between the conceptual overviews in the developer's guide, and the complex implementation required for a production-ready application.

What Is a Code Sample?

A code sample is a piece of syntactically correct and semantically useful code, written to illustrate the functionality and usage of an API or a developer tool.

The code sample provides a stepping stone between the conceptual overviews in the developer's guide, and the complex implementation required for a production-ready application.

This short-and-sweet code sample is from the documentation for the Google Maps Android API (*https://developers.google.com/maps/documentation/android/map*). I've quoted the textual instructions as well as the Java code sample used to illustrate them:

> *To set the type of a map, call the GoogleMap object's setMapType() method, passing one of the type constants defined in GoogleMap. For example, to display a satellite map:*

```
GoogleMap map;
...
// Sets the map type to be "hybrid"
map.setMapType(GoogleMap.MAP_TYPE_
HYBRID);
```

Notice a few things about the sample:

- It's short.
- It includes an ellipsis, letting developers know that more code is likely to be needed in the actual implementation.
- Apart from the ellipsis, the code is syntactically correct, down to the semicolons.
- The author assumes some pre-existing knowledge. In this case, it's assumed the reader is acquainted with the Java language, and with the GoogleMap class as the model of a map within an application.
- The sample nicely illustrates the textual instructions.
- The single variable in the code has a meaningful name, "map."
- Short though it is, the code still includes a comment (the words after the double slash).

Code samples may be illustrative snippets within a tutorial or getting-started guide, like the above example. Or they may comprise a fully working demo app, usually linked from the documentation and available for download.

Let's start with the case of the demo app, then move on to snippets.

A Fully Working App as a Getting-Started Tool

One of the main aims of a code sample is to get the developer started as painlessly as possible.

TTFHW (Time to First Hello World) is a hot topic amongst developers who use or build APIs. It's a bit like car enthusiasts talking about "0 to 100 in four seconds." For a happy getting-started experience, the trick is to get TTFHW as short as possible.

A Demo App Embedded into the Page

In some cases, a demo app may be very simple. For example, it's possible to demonstrate the Google Maps JavaScript API with a scribble of JavaScript (*https://developers.google.com/maps/documentation/javascript/tutorial#HelloWorld*).

The following Web page displays a map centered on Sydney, New South Wales, Australia:

```
<!DOCTYPE html>
<html>
 <head>
  <meta name="viewport"
        content="initial-scale=1.0,
user-scalable=no" />
  <style type="text/css">
   html { height: 100% }
   body { height: 100%; margin: 0;
padding: 0 }
   #map-canvas { height: 100% }
  </style>
  <script type="text/javascript"
   src="https://maps.googleapis.com/maps/
api/js?key=API_KEY
&sensor=SET_TO_TRUE_OR_FALSE">
   </script>
   <script type="text/javascript">
    function initialize() {
     var mapOptions = {
      center: new google.maps.LatLng(-
34.397, 150.644),
zoom: 8
     };
     var map = new google.maps.Map(
      document.getElementById("map-can-
vas"), mapOptions);
    }
    google.maps.event.addDomListener(window,
       'load',
       initialize);
   </script>
  </head>
  <body>
   <div id="map-canvas"/>
  </body>
</html>
```

A Downloadable Demo App

A fully working demo app may be fairly complex and the development project includes a number of files containing code, CSS, images, and other resources. The best thing is to store the app in a public repository and include a link in the documentation, so that developers can download the demo and import it into their development environment.

An example is the demo app supplied with the Google Maps Android API Utility Library (*https://github.com/googlemaps/android-maps-utils/tree/master/demo*). The setup guide for the library includes a description of the demo app, which is included as part of the downloadable library (*https://developers.google.com/maps/documentation/android/utility/setup#demo*).

Often the development team creates the demo app, or the technical writer has a lot of help in the development. Certainly the technical writer will review the code, paying particular attention to the comments, the naming of variables and methods, and the overall readability and simplicity of the sample.

Illustrative Code Snippets

The introductory section of this article included a very short code snippet. Let's look at some further examples.

Help Developers Understand the API

To help developers get started quickly, samples may illustrate the major features and primary use cases of the API. To a developer, a code sample means more than a block of prose.

A good example is the landing page of the Google Maps Android API documentation (*https://developers.google.com/maps/documentation/android/*). The page includes six medium-length code samples, starting with "Hello Map" and moving on to common, popular use cases such as adding a marker to a map and drawing a line from one location to another. A particularly nice feature of the page is that each code sample is accompanied by a video showing the code in action.

Illustrate a Specific Part of the API

Other samples aim to help developers grasp the purpose and usage of a specific feature, method, or function.

Let's take a look at the guide to marker clustering in the Google Maps Android API Utility Library (*https://developers.google.com/maps/documentation/android/utility/marker-clustering*). It begins with an explanation of the concept of marker clustering (a technique used to deal with a large number of place markers concentrated in a small area on a map, without making the map hard to read).

Then follows a very short step-by-step guide:

Here is a summary of the steps required:

1. *Implement ClusterItem to represent a marker on the map. The cluster item returns the position of the marker as a LatLng object.*
2. *Add a new ClusterManager to group the cluster items (markers) based on zoom level.*
3. *Set the map's OnCameraChangeListener() to the ClusterManager, since ClusterManager implements the listener.*
4. *If you want to add specific functionality in response to a marker click event, set the map's OnMarkerClickListener() to the ClusterManager, since ClusterManager implements the listener.*
5. *Feed the markers into the ClusterManager.*

Next up are the code samples, interspersed with very short textual introductions. Here's an example from the page:

In your map activity, add the ClusterManager and feed it the cluster items.

```
  private void setUpClusterer() {
    // Declare a variable for the cluster manager.
    private ClusterManager<MyItem> mClusterManager;

   // Position the map.
   getMap().moveCamera(CameraUpdateFactory.newLat
LngZoom(new LatLng(51.503186, -0.126446), 10));

   // Initialize the manager with the context and
the map.
   // (Activity extends context, so we can pass
'this' in the constructor.)
     mClusterManager = new ClusterManager<MyItem>
(this, getMap());

     // Point the map's listeners at the listeners
implemented by the cluster
     // manager.
     getMap().setOnCameraChangeListener(mCluster
Manager);
     getMap().setOnMarkerClickListener(mCluster
Manager);

     // Add cluster items (markers) to the cluster
manager.
     addItems();
  }
```

Points to note:

- The sample doesn't show any of the plumbing and boilerplate code that typically clutters a production-ready application, such as library import statements and routine error traps.
- The page includes the primary method that does the clustering (called setUpClusterer()) and a class that's important for developers to understand (public class MyItem implements ClusterItem).
- Distinct bits of processing are separated out into functions and given a meaningful name: setUpClusterer() and addItems(). This helps the reader grasp the concepts.
- There are plentiful code comments.

- Syntax highlighting (different colored text for each syntactical component) improves the readability of the code and makes developers feel right at home.

Summarizing the Goals of a Code Sample

A code sample should:

- Get the developer up and running as quickly as possible.
- In combination with the textual commentary, provide high-level, conceptual information that is not easy to find by reading the in-depth reference documentation or code.
- Be syntactically correct, and thus suitable for copying and pasting into an existing project or framework.
- Work (that is, do what it's meant to do) when dropped into a development project.
- Be as simple and clear as possible.

How to Create Sample Code

Here are 10 guidelines for creating a useful code sample:

1. Find out the primary use cases for the API or tool you're documenting. Those are what your code should illustrate. Talk to the development team and product managers and read the requirements or design document. If you're lucky, there'll already be some useful comments in the code, too.
2. Find out what you can safely assume about your audience. Are they are familiar with the technology you're working with (in my case, Java and Android development)?
3. Find some "real" code that does something similar to what you need: perhaps a similar function, or a working implementation that you can simplify for use as a sample. Or find a developer to help you write the sample from scratch.
4. Simplify the code. Remove functionality that's unique to your environment or that adds super-duper but superfluous features. For on-page code snippets, remove plumbing, such as import statements, and all but the essential error handling.
5. Make the code easy to read:
   - Make names meaningful. This includes the names of variables, classes, methods, and functions.
   - Keep methods and classes short. Compartmentalize logically related chunks of code into separate methods, classes, or functions, with meaningful names. This will help people grasp the concepts.
   - Pay attention to the indentation of the lines of code. It should follow a standard pattern and should illustrate the logical components (class and method definitions, method calls, and so on) of the code.
6. As far as possible, have all the necessary code in one piece, so that people can copy and paste easily.
7. Add code comments. Developers often add technical comments to the code, but don't include an introduction or conceptual overview of what's going on. And our target audience, also developers, will often read code comments where they don't read other documentation. So code comments are a great medium for technical communication!
8. Put your code into a framework and test it. The framework should be one that your audience will be using. In my case, that's an Android development project.
9. Take screenshots of the code in action. They're often useful to give people an idea of the goal of the sample code, and of what the API or tool can do.
10. Get the code reviewed by the development team.

Making Your Samples Super Useful

These are extra tips for increasing the usefulness of your code samples:

- Use the terms "foo" and "bar" in the sample. Developers associate these two terms with an example. Some people even search for "foo" and "bar" plus the specific tool or API, when looking for examples. (Thanks to Sergii Pylypenko and Ben Buchanan who pointed this out in a comment on Google+ [*https://plus.google.com/u/0/+SarahMaddox/posts/4oUpBs2CxAs*].)
- If your content management system supports it, put the code in a source repository and auto-include sections into the documentation using a "snippet" or "excerpt" tool. There are advantages to having your code all in one place, rather than scattered across multiple documents. This is especially useful if your code is a complete demo app. Bitbucket (*https://bitbucket.org/*) and GitHub (*https://github.com/*) are examples of online repositories. GitHub offers "gists" specifically for sharing code snippets (*https://help.github.com/articles/creating-gists*).
- If your code is a fully working demo app, hook it up to an automated test framework to help ensure the code samples remain compatible with changes to the API or tool that they document.

*This article quotes examples from the documentation for the Google Maps Android API (*https://developers.google.com/maps/documentation/android/*) and the Google Maps JavaScript API (*https://developers.google.com/maps/documentation/javascript/*). Thanks to Google for sharing this work (*https://developers.google.com/site-policies*), which is used according to terms described in the Creative Commons Attribution 3.0 License (*http://creativecommons.org/licenses/by/3.0/*). Note that the documentation is under continuous development, as the Google Maps APIs release new features and improvements. This means that the text or code may have changed since this article was written.*

SARAH MADDOX *is a technical writer in the Developer Relations team at Google. Before becoming a Googler, Sarah worked at Atlassian and various other companies in Australia, the UK, the Netherlands, and South Africa. With around 15 years of technical writing under her belt, and 9 as a software engineer, she has a history of persuading words and code to play nicely together. Sarah's blog is* ffeathers.wordpress.com.

shutterstock.com/Stuart Miles

How Much Programming Do You Need to Know to Write API Documentation?

By SAMUEL WRIGHT

JOBS IN THE DEVELOPER documentation space, such as API reference material, quick start guides, and programming examples, are on the increase. Which is all very well for writers who have moved over from working as a programmer, but how do technical writers without a formal programming background land these more technical roles? And how much programming do you really need to know before you can brag about your API documentation skills?

The bad news is, yes, you need do need to be able to read code to write useful API documentation. The good news is you do not need to become a programming expert (although that does help) and it is probably easier to learn than you think.

Picking a Programming Language

If you do decide to learn a programming language, the obvious one to start with would be the one in which the project you want to document is written. But even learning a different programming language will teach you concepts that you can reuse in other languages. So which programming language do you start with? Everyone has a different opinion here, so fall back on the best advice I've heard on the matter: "Pick the programming language you have most support for" (Jessica Rose, *Close to Clever*). Whether that is a friend, a colleague, or a buddy from a local programming language meet-up, having someone who can explain away the obstacles you are hitting in person makes a huge difference to how fast you can pick up a language.

Once you have picked a buddy and a language, ask him or her which online resources or books are best, and start working your way through those. Decide whether you prefer interactive tutorials or read-along guides, but either way make sure you actually write the code examples and run them because just reading the text won't get you very far. Likewise, figure out if you prefer paper books or ebooks, and use what works for you.

Here is a list of some common programming languages with descriptions quoted from Wikipedia and a couple of tutorials for each one. I've intentionally not mentioned C++ or Java, as I think you're better off learning another programming language before learning those.

Python

"Python is a widely used general-purpose, high-level programming language."

- *www.learnpython.org*
- *http://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_3/*

Ruby

"Ruby is a dynamic, reflective, object-oriented, general-purpose programming language."

- *http://ruby.learncodethehardway.org/book/*
- *http://mislav.uniqpath.com/poignant-guide/book/*

JavaScript

"JavaScript (JS) is a dynamic computer programming language."

- *http://learnjavascript.co.uk/*
- *www.codeavengers.com/javascript/17#1.1*

PHP

"PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language."

- *http://php.net/manual/en/tutorial.php*
- *http://devzone.zend.com/6/php-101-php-for-the-absolute-beginner/*

It is worth bearing in mind that reading code that someone else has written is almost always easier than writing your own code from scratch, although at some point in your career you are bound to come up against something that looks like it has been written to be intentionally incomprehensible.

Picking a Project

Once you have chosen a programming language and worked through some tutorials, you may think you know how this programming thing works. The way to find out whether you know how to program is to find something that you want to program, and *program* it.

The ideal project is something non-trivial but useful and of interest to you, and in practice that means you often end up "reinventing the wheel." Maybe it is a three-page blog, or maybe it is a command line program that lists all of the index markers in your XML documentation and downloads definitions for them from an online glossary. Whatever it is, you'll find the process of figuring out how to write it invaluable when it comes to knowing what type of information developers need.

A Note on Generated Documentation

Most programming languages have a (more or less formalized) method of documenting code using code

comments that are extracted and converted to HTML or PDF. When you are writing or editing this type of documentation, you'll edit the source code files directly, so find out what format the comments are written in (often but not always a variation of Markdown).

The important thing to be aware of here is that code comment-generated documentation does *not* mean that documentation writes itself; it means that the documentation compiler takes care of the easy mindless stuff and lets you make sure the text actually makes sense and is useful for the programmer reading the documentation.

JSDoc is one way to document JavaScript code. The following example shows a function definition for a function called Book (the last three lines), with documentation in JSDoc format code comments (the first six lines).

```
/**
 * Represents a book.
 * @constructor
 * @param {string} title - The title of the book.
 * @param {string} author - The author of the book
 */
function Book(title, author) {
  // This function does not do anything yet.
}
```

We'll look at what sort of information we need to add to this documentation in the next section.

Considerations when Documenting APIs

So you have learned the basics of your chosen programming language and have an API to document. What now? When writing API documentation, what you need to keep in mind is that the developer can often read exactly the same code that you are reading, so what your documentation needs to do is add value on top of what can be easily gleaned from the code.

The previous JSDoc description of the Book function omits the following potentially useful information (this is not a complete list):

- What is the maximum length of the title and books strings?
- Is a one character title or author valid?
- What happens if you call the function with an empty string in place of the title or author?
- What other restrictions on the parameters are there? What happens if you call the function with values outside these restrictions?
- Does the function return a value? What type is it?
- Does the encoding of the string matter?

None of these issues (for this admittedly absurdly simple example) are at all specific to JavaScript, but the answer to all of them *should* be in the source code you are documenting. All you need to do is read it, using your newly acquired programming literacy.

Writing Good Code Examples

Good developer documentation is full of code examples, but you can't just copy and paste code from a random file and expect it to illustrate your point! Each example should illustrate one syntax element or API endpoint, or be part of a longer ongoing example that you gradually build up. Avoid using code features that you have not yet explained, "cunning tricks" to save space, or complex optimizations—in code examples, clarity is king.

And always, *always* test your code examples! There is nothing quite so frustrating as copy and pasting a code example into a file, running it, and finding out that the author has not double and tripled checked the code. It is best to set this up so it happens automatically, that way you don't have to worry about forgetting about it. Exactly how you go about this depends a lot on your authoring and programming environment, but a little extra effort upfront will save you a lot of headache and manual testing later.

Learning More

One downside of learning to program is that you will find it is like pulling at the end of a very large ball of yarn. There is an incredible amount of detail to learn, and an almost infinite list of websites saying they can help you learn it! Here are a few of the better ones:

- Learnable: An online learning environment providing access to online books, courses and video lessons focused on web design and web development. (Subscription based).
- Udemy: Online courses on *almost anything*, including programming (Subscription based).
- Coursera: Over 400 free online courses provided by universities worldwide.
- Stack Overflow: Ask and answer questions about programming topics -- for when you can't find the answer anywhere else.

You can also subscribe to mailing lists and IRC chat rooms, use Twitter hashtags, look for local meet-ups, and follow blogs.

A note on learning resources: as helpful as the people who write Stack Overflow questions and answers, blog posts, and tutorials are, don't assume they are experts in their field, and don't take answers as an indication of best practices. Yes, they might answer your immediate problem, or show you how to get a step closer to a solution, but do take anything you find on the internet with a grain of salt.

Doubts?

If you are still in any doubt, next time you see an interesting writing job that requires programming experience, remember which language they wanted. Look it up and try to learn it, and you might be pleasantly surprised. If nothing else, you can use the experience as an example next time someone at an interview asks you for a recent challenge! i

SAMUEL WRIGHT (@plaindocs) *is a freelance technical writer, specializing in the minefield that is API and developer documentation. After working in various parts of the world, Sam chooses to live in Berlin, where he initially moved to work with Nokia. At Plaindocs he detangles API documentation and implements documentation systems and processes.*

International Summit Awards Best of Show and Distinguished Winners Announced

BY NATHANIEL LIM | *Associate Fellow*

AT THE SUMMIT AWARDS Banquet in Phoenix on 20 May, STC presented the award for Best of Show in the 2014 International Summit Awards (ISA) for Research Priorities for Tropical Ecosystems Under Climate Change. Judy Wyrick accepted the award on behalf of Jeffrey Chambers, Richard Norby, and the Biological and Environmental Research Information System.

This year's Best of Show (BOS) judges, Louellen Coker—Lead, David Dick, and Thea Teich, reviewed three Distinguished Technical Communication Award winners that were nominated for Best of Show by ISA judge teams. Based on the ISA judges' recommendations and their own reviews of the entries, the BOS team praised the winner with this citation:

For providing a connected, thorough, and eminently understandable explanation of a complicated but universally important subject to a diverse audience.

STC also recognized 5 Distinguished Technical Communication Award winners, two of whom were at the Summit Awards Banquet.

2010-2011 ICCVAM Biennial Progress Report entered by Catherine Sprankle, Donna Jeanne Corcoran, and the National Toxicology Program Interagency Center for the Evaluation of Alternative Toxicological Methods.

2012 ALCF Annual Report, entered by Renee H. Carlson, James Collins, and the ALCF Communications Team. Ms. Carlson accepted in person.

Informal Learning Basics, entered by Saul Carliner, Stephanie Castellano, and Mark Morrow.

The Pocket Paper Engineer, How to Make Pop-Ups Step-by-Step, Volume 3, entered by Carol Barton and Eleni Smitham.

Welch Allyn CP150/250 Office and Hospital Carts, Assembly Instructions, entered by Bobbi Werner and Corrie Baum. Ms. Werner accepted in person.

These 6 entries were on display in the exhibit hall during the Summit.

Mr. Carliner describes his experience since winning his award:

> The feedback was helpful. It validated decisions that my editors and I made (we had an unusually intense copyediting round just before going to press), and I know that the award brightened the days of my acquisitions and development editors.
>
> Furthermore, the specific feedback from the judges was also complimentary enough that if I ever need a bit of a boost, I'll re-read the comments.
>
> Although I am not planning to revise this book anytime soon, I am currently revising another book in the series and will review the feedback as I go.

Ms. Werner describes her experience since winning her award:

> My manager nominated me for a Level II Quality Award at my company. In fact, she nominated all of my Tech Comm colleagues who won awards at the regional and international competitions: two writers and three illustrators.
>
> All five nominees just received this award from our President and CEO. His letter to each of us states the following:
>
> "I want to personally thank you for your contributions in creating Welch Allyn's Technical Documentation for our customers.... As a result of your contributions, you have ensured that Welch Allyn's customers have clear and concise technical documentation for our products. Congratulations on the awards from the Society for Technical Communication (STC)! This type of initiative and commitment among Welch Allyn employees makes me very confident that we can build on our past successes as we look to a challenging future.
>
> "Keep up the great work!"
>
> In addition, we will share the feedback received from our reviewers to mentor junior writers and promote participation in future competitions.
>
> Each one of us received a $1000 cash award as well. In today's economy, that's a most welcome show of appreciation for the excellent work we do.

This year there were 62 entries, with 44 winners or 71% receiving awards. Thank you to the 2013–2014 International Summit Awards committee members: training manager, Dorothy McDuffie; entries manager, Elizabeth Bailey; judge manager, Paula Robertson. Thanks also to the local competition managers and judges, ISA competition judges, competition entrants, and STC staff liaisons Kobla Fiagbedzi and Lloyd Tucker.

Congratulations again to the 2014 ISA Best of Show and Distinguished Technical Communication Award winners.

2015 Membership Season is Open!

THE STC MEMBERSHIP SEASON is now open! Renew your membership today and continue setting yourself apart from your peers by discovering even more of what STC has to offer in 2015.

2015 memberships are even more valuable:

1. A limited-time **Early Renewal Discount** is now available for members. Save $30 on dues during the early renewal period. This brings a Classic membership down to $195, but you must renew early to get this discount! Use Promo code: **STC2015.**
2. Back by popular demand, **Gold Value Package members receive printed copies of *Intercom*.** Ten issues of *Intercom* magazine will now show up in Gold member mailboxes this year—without an increase in dues! Gold Value membership still includes: all Classic member benefits, one chapter, all SIGs, five free Webinars, half off one online course, and Summit Really Early Bird pricing anytime. The cost will remain $425 for the year, which is a real bargain with all the bundled benefits.
3. **The 2014 *Salary Database* is being published soon!** All 2015 STC members will continue to receive a free copy of STC's award-winning *Salary Database*. As you may know, STC's *Salary Database* is a tool that can be used to conduct more powerful job searches, make a strong case for a raise, or prepare department payroll budgets.
4. **More continuing professional development discounts and free education** will be available for STC members. Stay a step ahead of your colleagues by attending cutting-edge, content-rich webinars, virtual courses, STC's Summit, and virtual conferences. Don't forget about 24/7 free access to 140 on-demand seminars.
5. **New special interest group launched—the STC Agile SIG.** Connect with like-minded professionals focused on topics specific to Agile. This and all STC virtual communities are available to you day or night.

All memberships continue to receive online subscriptions to the award-winning *Intercom* and *Technical Communication*; access to the Salary Database; our library of 140 on-demand seminars; deals from Hewlett Packard, LifeLock, MadCap, Lenovo, ComponentOne, and others with our Affinity Program; significant discounts on STC webinars, virtual courses, recorded webinars, and the annual Summit; a 14-day head start on all jobs posted to the STC Job Bank; and networking opportunities across the globe.

Renew your STC membership today and be a part of *your* growing Society. Visit *www.stc.org/membership/join-or-renew-now* for more information, and email *membership@stc.org* with any questions. We look forward to a great 2015 for technical communication!

Reminder of Deadlines for Awards and Honors

THE DEADLINES FOR NOMINATIONS for many of STC's awards and honors are upcoming. Please see the STC website, *www.stc.org*, for more information or to find out how to nominate someone.

- Associate Fellow Recommendations: 1 October
- Fellow Nominations: 15 October
- Sigma Tau Chi and Alpha Sigma Honors Societies: 26 October
- Jay R. Gould Award for Excellence in Teaching Technical Communication: 2 November
- Ken Rainey Award for Excellence in Research: 2 November
- Distinguished Community Service Awards: 2 November
- Community Achievement Awards: 27 January 2015
- Community Pacesetter Awards: 25 March 2015
- STC Election: 9-20 March 2015

STC Summit 2015: Call for Proposals Open

THE CALL FOR PROPOSALS for the Technical Communication Summit 2015 will open on 18 August 2014 and close on 12 September 2014 at 5 PM EDT (GMT-4). We invite you to contribute your ideas, studies, experience, and expertise at the Summit, which will be held 21-24 June 2015 in Columbus, Ohio, USA.

Who attends the Summit?

Our attendees are professional technical communicators at all levels, from beginners to seasoned veterans. This includes technical writers, editors, illustrators, managers, information designers, architects, content strategists, instructional designers, usability and user-centered design practitioners, researchers, and students. Our attendees represent a wide range of industries, including health care, manufacturing, information technology, education, art, science, finance, telecommunications, software development, and transportation.

Who can submit a presentation proposal?

Anyone with experience in the latest communication techniques, technologies, approaches, and solutions is invited to submit. You are not required to be an STC member to submit a proposal.

What types of topics is the Program Committee seeking?

After the Call for Proposals closes, the Conference Chair and Program Managers will sort the proposals into tracks to be reviewed. The Program Committee has renamed the tracks for the 2015 Summit as follows:

- Art, Design, and Visual Communication
- Leadership and Management
- Writing and Communication
- Tools and Technology
- Training and Research

Track assignments depend on the topic and keywords provided by the submitter, as well as patterns and trends observed throughout all of the submissions. The Call for Proposals website provides more specific topic examples for each of these tracks. In addition, the website also describes the types of sessions that can be proposed, so please consider the session type that best fits your topic and your presentation style.

How are proposals evaluated?

Our program committee considers the following when reviewing proposals:

- Is the topic timely and of interest to our attendees?
- Is the topic important for our industry?
- Does the abstract grab the reviewer's attention and make an immediate case for reading more about the session?
- Does the session proposal demonstrate a solid direction and purpose for the session?
- Is the proposal clear, concise, and well-written?

What is the schedule for the Call for Proposals?

- 18 August 2014: Call for Proposals opens
- 12 September 2014: Call for Proposals closes at 5:00 PM EDT (GMT-4)
- Mid-November 2014: Submitters are notified of proposal status
- Early December 2014: Early-bird registration opens for the Summit

If you have any questions, please contact the Program Advisory Committee or Conference Committee (*stcsummit@gmail.com*) for more information. Visit *http://summit.stc.org/call-for-proposals/* for more information and to submit a proposal.

Did You Miss the Summit?

Don't miss this second chance at attending the Summit—from the comfort of your home or office!

You can "attend" all the 2014 sessions that interest you . . . at a time and place of your choosing and without having to travel.

Summit@aClick lets you attend the conference you missed! Content of almost every session was captured (audio and visuals) and will be available in the STC Live Learning Center. Members who did not attend the Summit can purchase Summit@aClick now—see the STC website for details! **STC is the only meeting in the technical communication field that offers this ability to attend the conference sessions without having to travel.**

Note: Summit@aClick was included with the full registration fee for the Summit and Virtual Track. Attendees will receive an email with username and password roughly 6-8 weeks after the Summit.

Summit@aClick

Holiday Madness: Tough Totes Shoulders (Un)Successful Startup

shutterstock.com/Marques

BY HEIDI L. EVERETT | *Guest Columnist*

BLAIRE MATTHEWS WORKED full time on a manufacturing line, but he was known locally for the high-quality, hand-crafted leather goods he made as a hobby. People referred to his computer bags and purses as "tough totes" because of their rugged look and durability.

Family and friends who had received bags as gifts encouraged Blaire to branch out and sell his wares via the Internet, especially before the next Christmas shopping season. After much convincing, that's what he did.

In February, Blaire established Tough Totes LLC and filed for a trademark. Throughout the spring, he created samples of eight leather bag designs he would use as his initial product offering at price points ranging from $100 to $300. Blaire also purchased the raw goods needed to create a small inventory, including leather, fabrics, buckles, and snaps. He enlisted the help of a friend who is an amateur photographer and writer. The two snapped lifestyle shots of the Tough Totes and wrote descriptions of each bag.

In May, Blaire hired BUZZ, an agency specializing in branding and marketing. The agency's job was to create a company identity for Tough Totes, to develop a manageable e-commerce website, and to advise and assist Blaire on a sales and marketing strategy for the business launch.

The website BUZZ developed was relatively small, but attractive and easy to use; more importantly, it was created so that Blaire could make basic updates himself. The pages showcased the eight products with photos, product descriptions, and specifications. In addition, the site celebrated the fact that Tough Totes were handcrafted and not mass-produced. Financial transaction capabilities were incorporated into the site by embedding the functionality of a well-known e-commerce business that processes payments through the Internet. Upon payment, Blaire would receive an email notification of the order. The website also included a customer contact function that generated emails to Blaire.

During the online order process, customers would receive information to manage their expectations of the experience. For example, prior to clicking the submit button, customers would be reminded again that each bag is handmade and takes up to two weeks to create before shipping. The notification also stated that customers would receive an email when their order shipped. Finally, the notification clarified the name Blaire Matthews—and not Tough

This column features ethics scenarios and issues that may affect technical communicators in the many aspects of their jobs. If you have a possible solution to a scenario, your own case, or feedback in general, please contact Derek G. Ross at *dgr0003@auburn.edu.*

Totes—would appear on their credit card statement as the vendor.

BUZZ recommended a four-pronged approach for the Tough Totes sales and marketing launch strategy that included:

- A public relations blitz through relevant trade and lifestyle media
- Sales through online marketplaces like Amazon and Etsy
- Targeted ads on social media, music, and search engine sites
- Social media spaces on Twitter and Facebook to announce new product launches, share customer testimonials, and generate a buzz about the company and its products.

The Tough Totes website, Facebook, and Twitter accounts went live to coincide with the public relations blitz in September. Paid advertising started in October. Blaire and a friend were hard at work creating a stockpile of bags in anticipation of the holiday buying season.

As bags were sold and shipped, Blaire included a note card in the package that reinforced the brand attributes of being high quality and handcrafted. The card thanked the customer for sharing a love of leather goods and supporting small business. It outlined steps for returning the product if the customer was not completely satisfied. Finally, the card encouraged customers to share their thoughts about their purchase on the company's social media sites.

As hoped, customer reviews were extremely positive, and Blaire was pleasantly surprised by the number of customers who shared reviews online. Business began to pick up.

By early November, Blaire called upon family and friends to "earn a little extra cash for the holidays." He asked them to help in any way they could after work and on weekends, from placing orders for more raw goods, to sewing the linings, to attaching buckles and packaging shipments. Blaire even took vacation days from his full-time job to make bags. Tough Totes was having trouble keeping up with orders; production time was estimated at three to four weeks rather than the two weeks noted on the website. The company continued to promote the product online and accept orders and payment.

The second week of December, Blaire posted the following message on Tough Totes social media sites, "The holiday demand highly exceeded our expectations. We're doing everything we can to fulfill orders on time." He continued to fulfill orders in the order in which they were received. In addition, the Tough Totes website continued to take orders and payments without adjusting the estimated time of delivery notice. Blaire was confident the order volume would soon level out and decrease, and that all orders would be fulfilled. He also didn't want to miss out on additional sales.

On 1 January, customers began flooding Blaire's email inbox from the customer comments section of the website; worse yet, customers began posting negative reviews on Tough Totes social media sites, many asking about the status of orders that had not been fulfilled. One customer posted several weeks in a row that an order had been placed 29 November for $180; the bag had yet to be delivered or money reimbursed. Blaire was quick to respond to social media posts, apologizing for not meeting order delivery expectations. He directed customers to provide their order confirmation number, promising that the situation would be remedied as quickly as possible. However, Blaire did not follow up through email or phone with a new product delivery date. Reimbursements were not being offered because the money had been invested in raw goods-and-labor costs. The second week of January, Blaire started deleting customer posts from Tough Totes social media sites; he also stopped checking his inbox because the volume of emails was overwhelming. He focused his efforts on getting bags produced and shipped.

The Tough Totes website continued to accept orders and payments into February. At the same time, Tough Totes social media sites read, "We're doing everything we can to make things right after the holiday demand exceeded our expectations." By March 1—after less than a year in business—Blaire deleted the Tough Totes website and social media presence. More than 100 orders were not fulfilled; money was not returned.

Editorial Note

With the holiday season fast approaching, many of us will have our hands full trying to meet the needs and expectations brought about by increased consumer demand. We may be working with organizations that offer design and consulting services for small businesses, and may even be designing websites and formulating marketing strategies for startups like the one featured in this month's column. As we do, we may be asked to create something that potentially exceeds our clients' abilities. What ethical responsibilities do our skills require?

In our last column we considered the problem of ableist language—language that casts disability as inherently negative. In doing so, we recognized once again the power that our language use has in shaping action and perception. Here, we face a related issue—what happens when we apply our craft so well that we create something our audience can't handle? Can we even consider our craft well-applied if a client ultimately loses control of their business as a result?

As always, we welcome your responses. Let us know your answers to the questions we've posed, your thoughts on our roles as designers and consultants in general, or send us your own ethics cases or column ideas. Please send your responses to derek.ross@auburn.edu. Responses will be printed in an upcoming issue of Intercom *as space permits.*

—Derek G. Ross, Column Editor

Serving Small Business

Tough Totes boom-and-bust case is not unique to small businesses or even larger, more established firms. In fact, as recent as Christmas 2013 major retailers Amazon.com, Kohl's, Wal-Mart, and 1-800 Flowers, along with distributors UPS and FedEx, were railed in mainstream and social media by consumers who had not received their purchases in time for the holidays. Coverage in *U.S. News, USA Today, New York Daily News,* and ABC.com cite 15% to 17% increases in online shopping coupled with last minute delivery guarantees as key contributors to consumer demands that exceeded forecasts and capacity. While big-name retailers are often in a position to quell angry consumers by offering gift cards or full reimbursement, small businesses like Tough Totes may not be, causing a devastating blow to the organization.

According to the Small Business Administration (2014), in 2011 the United States was home to 28.2 million small businesses. In 2011, just over 409,000 new businesses started while 470,736 closed. Approximately 50% of all new businesses will survive the first five years of operation. More than 75% of small businesses do not have employees. As a result, multi-tasking owners may rely on their family members and friends for assistance and advice. Further, they may enlist the services of industry experts, like BUZZ.

Questions

Does BUZZ bear any responsibility for Tough Totes' eventual failure?

What role should technical communicators perform for small businesses, particular technical communicators who may work in an agency like BUZZ?

Is it the ethical responsibility of a design company to be broader business consultants who clearly outline the opportunities and risks of the marketing strategies and technology deployments they facilitate?

Or, should their advice be limited to technology deployment and content development?

HEIDI L. EVERETT *is senior writer at Vantage Production, a United Communications Group company, and an adjunct faculty member at St. Cloud State University. She is completing her doctorate in technical communication and rhetoric at Texas Tech University.*

REFERENCES

Small Business Administration. Office of Advocacy. Frequently Asked Questions. 2014, *www.sba.gov/sites/default/files/FAQ_March_2014_0.pdf.*

MK Graphics

Technical Illustrations

Page Layout

Press Sheet Inspection

Design

Business Cards

Letterhead

mkgraphics53@gmail.com

Communities of Practice for the Workplace Environment

shutterstock.com/Monkey Business Images

BY ALANA BAKER | *Guest Columnist*

EVERY WORKPLACE has a mission. With qualified employees who share skills and a desire to achieve that goal, workplace environments already possess the key to garnering success for the institution and for the individuals. The mission is only sustainable, however, when communication is open and fostered through a team effort.

No matter how many clients are involved, employees manage to become absorbed in office politics: self-interests and countless other games that have nothing to do with the mission of the organization. The results of such may include decreases in several areas, including employee retention rates and profit. If the situation is left unchecked, in the end, only an empty building will remain.

How do I know this? I was once a staff member in such a dysfunctional office environment before I became a manager elsewhere. As a staff member, it became increasingly more apparent every day that client interests were fading into the background of managerial focus. I found that outlets for me to cultivate my own knowledge and respond to clients also grew more limited. There is hope, however, when the interest remains organization-centered: that hope is focusing on the community of practice.

Communication

Part of being an effective team and accomplishing tasks is having a common goal or shared group interest based on common knowledge. Then, the team must find a way to use skills to reach that common goal. For example, for an organization seeking to increase its presence in the media, one must consider which employees have the Web design, public relations, and networking skills necessary to accomplish this task. This is easier said than done, however, as self-interests can become a problem in such scenarios, for such self-interest can prevent individuals from recognizing the skills they have in order to accomplish tasks. In my case, it was a manager's personal desire to obtain authority that conflicted with the team's objective, thus hindering communication throughout the group. This situation could have been remedied if the manager had

This column features the work of individuals currently enrolled in or recently graduated from educational programs in the field. Contributors examine how theories and concepts encountered in their classes can be applied to technical communication practices. To submit a column, email the editor at *STAMANTK@ecu.edu.*

been more open to sharing ideas and communicating instead of focusing on self-interests (i.e., advancing his or her status in the organization).

Self-interests, when they hinder this realization, disrupt the most important factor for the success of any workplace environment: communication. Communication promotes the sharing of knowledge and ideas—sharing that fosters self-realization of skills and team-realization of how these skills can be applied to completing tasks that will help the organization fulfill its mission.

In my case, the manager's decision to limit my ability to share knowledge with administration resulted in key tasks being outsourced, which led to unnecessary spending. If I had been allowed to communicate and use my knowledge, the organization could have saved a substantial amount of money and could have prevented some layoffs. This scenario demonstrates how one incident can create a ripple effect that impacts the entire organization. Because of that manager's self-interests, money and employees were lost. Upon becoming a manager, I vowed to be different, better—and it was successful. The key to this success was taking a community of practice approach to the management process. The following sections of this paragraph will define the community of practice model and outline requirements for growth and development in the workplace.

The Community of Practice

The best model of a workplace team, in my own experience as a manager, is the community of practice, a concept Jean Lave and Etienne Wenger (1991) originally coined as a learning theory. A community of practice is a group of individuals with a shared interest and shared knowledge base who possess skills and apply them to achieve a specific goal. This shared knowledge base is known as "domain," or the specific kind of knowledge required to carry out a task. For example, the domain of a group of Web developers might include knowledge about particular platforms or software. They must use shared knowledge to strengthen their domain to accomplish tasks and fulfill the mission.

When a group is given a task that relates to the overall mission, the community of practice emerges from a desire to increase domain in order to accomplish the task. The community of practice, therefore, is not created or established, but is a product of the desire to complete a task and domain knowledge in a group.

Each individual under my supervision (i.e., in my community of practice for a workplace environment) had a unique voice and special skills. As the manager, it was my role to recognize these skills and offer opportunities for members and myself to share with one another to accomplish tasks and achieve our mission. For example, I found it important for my staff members to meet once a week as a group to discuss areas that needed improvement and areas in which we excelled. Weekly data and observations were shared with the group, and each individual was given an opportunity to share thoughts at the meeting. By taking the time to analyze the data as a group, we were able to discover and develop new ways to improve from one week to the next. This approach mirrors the community of practice in that the members of the group united because of a shared interest in achieving the goal and applying domain knowledge.

Also, in sharing ideas, the group increased the domain knowledge as related to accomplishing the tasks leading to the goal. Lines of communication were open, and there was mutual trust. The group also had other opportunities to meet on a regular basis both face-to-face and electronically, formally and informally. When people are comfortable in a group and feel valued, they will share knowledge and passion.

Basic Tenets

In a community of practice, three tenets are at work:

- *Value.* Value is domain knowledge and the ability to apply this knowledge.
- *Language.* Language is the terminology used for processes in the community of practice.
- *Technology.* Technology refers to tools needed to perform the tasks in the group.

These three tenets are knowledge-based and hinge on communication. Value, language, and technology cannot exist alone, for each act as checks and balances for the other. For example, if there is no common value system of knowledge, then group members will have nothing about which to communicate.

To return to the previous illustration of an organization working to achieve media exposure, employees who exhibit skills in Web design, public relations, and networking must have knowledge about the organization in context of their own skill areas (i.e., value), the terminology for processes in the organization (i.e., language), and tools such as software used by the organization (i.e., technology) to carry out the mission. If software suddenly disappeared, then the community of practice would dissolve, because without technology, language and value of the system could not be shared.

In the same scenario, if a Web designer was unfamiliar with writing code for an encryption on the company's website necessary to launch the website, then this would destroy value; language and technology would be defunct. If any individual were unfamiliar with the mission of the company and policies, there would be no common language for the group, thus leaving no room for value or technology. Therefore, without one of the three tenets present, the community of practice dissolves.

The Mission Statement

In the community of practice, value, language, and technology are all under the umbrella of the shared interest of the group. In any workplace environment, the shared

interest or goal should be clear—it is outlined in a mission statement available to all employees and serves as the core of every action performed in the organization. With a mission statement, employees understand the purpose and identity of the organization, where it originated, and how the organization would expect employees to behave as a part of the entity. The mission statement unifies employees and states the shared interest that makes the community of practice viable in a workplace environment. A mission statement will help employees to accomplish tasks by allowing value, language, and technology to be communicated.

An analysis of an actual mission statement is helpful in illustrating the promotion of a shared interest in an organization. For example, Meredith Corporation's mission statement, found on their website, reads:

> For over 100 years, Meredith brands have been committed to providing women with information and inspiration to create a rich and meaningful life. By focusing on the core passions of family, home and self, we continue to play a vital role in the lives of 100 million women. Our influence extends to every life stage and spans every medium. Online and offline, we deliver quality content from the brands she trusts, across the subject matter that's most important to her. We create custom marketing solutions evolving from deep, strategic insights. Our expertise across media platforms and our ability to help clients build strong consumer relationships has made Meredith the leading media and marketing company focused on women.

Meredith's mission statement provides a brief description of the organization's purpose and history in the very first sentence. This is the shared goal of the company. The organization identifies its target audience as women and explains its goal of serving the target audience through multimodal channels, which are the technologies that employees must use to achieve the goal. In addition, employee commitment to serving its clientele is explained in the final two sentences, and language about the field must be used to make this communication successful.

By making its mission statement available online, Meredith ensures that its employees always have access to the mission and are, therefore, completely knowledgeable of and held accountable for company policy.

It is up to the manager to make sure that the mission is iterated and that lines of communication are open. If a manager is able to promote communication with and between employees and uses the mission statement to keep them motivated, communities of practice will grow, which will lead to the accomplishment of tasks and the overall mission of the organization. A manager may choose to institute mandatory team meetings and begin each meeting with a "round-robin" approach to sharing thoughts about the previous week. Sharing in this fashion should not be limited to staff members; the manager should be an active participant in the discussion. By participating in the meeting in the same capacity as staff members, the manager shows that he or she is part of the staff's community of practice and not simply a facilitator. Joining the staff in this activity lets the group know that the manager is willing to share ideas as well as listen to feedback. Individuals will not communicate without trust, especially when they cannot trust a manager, as was the case when I worked in a toxic environment. The mission statement, as shown in the Meredith Corporation example, can serve as a reminder for managers and employees of what the purpose is of the organization, holding both accountable for the future.

Conclusion

The community of practice is a system that can foster success because it allows groups to share knowledge and communicate through value, language, and technology. Success occurs when that shared interest is achieved and maintained, increasing value where knowledge is shared across the group.

In encouraging and contributing to the community of practice that formed in my office, there were few conflicts because communication was clear and open. Not one employee resigned based on dissatisfaction with the environment, and each year we reached assessment goals that accompanied our overall mission of service.

A manager must ensure the presence of knowledge pertaining to the mission and iterate this shared goal. The manager must note and adjust elements of the working environment in order to allow the community of practice to form and grow—if a manager is effective and devoted to the mission, the community of practice will undoubtedly develop. Workplace environments can certainly benefit from the community of practice in adding value to all parties involved. i

ALANA BAKER *is pursuing her doctorate in rhetoric, writing, and professional communication at East Carolina University where she is an instructor of English and an associate editor for* Technical Communication Quarterly. *Her research interests include exploring intersections in composition theory, creative writing theory, and technical communication theories as they apply to writing pedagogy; examining written interactions between cross-cultural communities; and applying rhetorical theories of ethos in pop culture. You can email Alana at* bakeral13@students.ecu.edu *or at* alanabaker336@gmail.com.

REFERENCES

Lave, Jean, and Etienne Wenger. *Situated Learning: Legitimate Peripheral Participation*. Cambridge, MA: Cambridge University Press, 1991.

Meredith Corporation. Mission Statement. In *Meredith*. Retrieved 9 December 2013, *www.meredith.com*.

Mark Your Calendar

Organization Events Across the Globe

10

F.Y.I. lists information about nonprofit ventures only. Please send information to *intercom@stc.org.*

1 25 Sept
The **STC Academic SIG** hosts a free preconference from noon to 5:00 PM immediately before the annual CPTSC and ACM SIGDOC conferences (listed below) in Colorado Springs, CO. For more information, visit:
www.stc-academic.org/

2 25-27 Sept
The **Council for Programs in Technical and Scientific Communication (CPTSC)** hosts Contexts and Connections: Considering Situational Factors Affecting the Structure and Nature of Programs at the University of Colorado at Colorado Springs in Colorado Springs, CO. For more information, visit:
www.cptsc.org

3 27-28 Sept
The **Association for Computing Machinery (ACM) Special Interest Group on Design of Communication (SIGDOC)** conference will be held at the University of Colorado at Colorado Springs in Colorado Springs, CO.
http://sigdoc.acm.org/

4 8–11 Oct
The **American Medical Writers Association (AMWA)** will hold its 74th annual conference, with a theme of "Shake, Rattle, and Write," at the Memphis Cook Convention Center in Memphis, TN. For more information, contact:
AMWA
+1 (301) 294-5303
amwa@amwa.org
www.amwa.org/events_annual_conference

5 13–15 Oct
The **IEEE Professional Communication Society (PCS)** hosts IPCC2014, Meeting Grounds: Lifelong Learning, from Classroom to Workplace, at Carnegie Mellon University in Pittsburgh, PA. For more information, contact:
IEEE PCS
http://pcs.ieee.org/ipcc2014/

6 26–29 Oct
The **Public Relations Society of America (PRSA)** hosts their international conference, "Leading the Way: A Fearless Future for PR," at the Washington Marriott Wardman Park Hotel in Washington, DC. For more information, contact:
PRSA
+1 (800) 350-0111
www.prsa.org/Conferences/InternationalConference/

7 27–31 Oct
The **Human Factors and Ergonomics Society (HFES)** will hold its 2014 international annual meeting at the Hyatt Regency Chicago in Chicago, IL. For more information, contact:
HFES
+1 (310) 394-1811
info@hfes.org
www.hfes.org/web/hfesmeetings/2014annualmeeting.html

8 31 Oct–4 Nov
The **American Society for Information Science and Technology (ASIS&T)** will hold its Annual Meeting, with a theme of "Connecting Collections, Cultures, and Communities," at the Sheraton Seattle Hotel, Seattle, WA. For more information, contact:
ASIS&T
asis@asis.org
www.asis.org/asist2014/

9 5–8 Nov
The **American Translators Association (ATA)** will hold its 55th annual conference at the Sheraton Chicago in Chicago, IL. For more information, contact:
ATA
+1 (703) 683-6100
ata@atanet.org
www.atanet.org/conf/2014

10 5–6 December
The **India Chapter of STC** will hold its 15th annual conference at the Vivanta by Taj hotel in Bangalore, India. For more information, contact:
STC India
www.stc-india.org/

* STC-related event

Bill Owen Coggin, 1948-2014

BY BILL LEAVITT | *Fellow*

BILL COGGIN, STC FELLOW and student advisor of the Bowling Green State University Student STC Chapter, passed away on 9 February 2014 after a long illness. Bill served the technical communication profession and STC both in the academic and professional/industrial arenas with significant personal and professional success.

Bill taught technical communication classes at Bowling Green State University (BGSU) and was instrumental in creating the BGSU English Department's technical and scientific communication program, which offered both Bachelor's and Master's degrees. Bill also created a large and successful student chapter at BGSU and served as its advisor and editor of the national students' STC journal. To help his students with their job searches, he established a fund that helps students attend and present papers at STC international conferences; this fund will continue to help students for the foreseeable future. Bill was a very popular teacher and student chapter advisor, and he did everything possible to help his students to successful careers.

At the Society level, Bill was elected region 5 director-sponsor for the term 1986–1989 and was nominated for second vice president of STC in 1989 and 1990. During my administration (1989–1990), Bill was appointed to be the first person to serve in the newly created position of assistant to the president for academic affairs, which was the principal contact for students and student chapters to be represented on the STC Board of Directors. In this position, he created an academic support role that has endured in one form or another for 25 years.

Bill has authored and co-authored many books and journal articles, and has conducted workshops and made presentations at academic and industry conferences, including those of STC. Beginning in the late 1990s, Bill made a number of trips to China to teach in BGSU's faculty exchange program at Xi'an Foreign Languages University, combining teaching English with learning Mandarin and doing research into Chinese education and culture.

He was born in Malvern, Arkansas, and received his BA and MA degrees from Louisiana Tech University. During this time, he met his wife, Betty, and they had two sons, Robert and Martin. He taught at Oklahoma State University (OSU) where he earned his PhD in Anglo Saxon literature and history. While teaching at OSU, Bill became a friends with Tom Warren, Professor Emeritus at OSU (also STC Fellow, Jay R. Gould Award recipient, and long-time STC leader). "Bill, one of the first graduate students to teach technical communication at OSU, not only was a compassionate teacher, he was also highly creative, coming up with novel ways to help students understand the importance of clear communication. Even though his dissertation was in literature, he became a strong advocate for technical communication, which earned him his first teaching job," Warren said.

Bill then taught technical communication at Miami University before spending the rest of his career at BGSU. Bill was the first in his family to graduate from high school; he worked in the oil fields to earn his way through college. He was drafted into the U.S. Army, where he served as a Vietnam interrogator and interpreter.

Bill Coggin and I go back to 1983, when we met during an STC conference. A small group of us were talking about STC and telling jokes. Bill and I became instant friends, as we shared a similar sense of humor and a desire to help STC members and especially students. Bill was student advisor at the BGSU Student Chapter during the time that I served as director-sponsor for STC region 5 (1983–1986). I feel that I trained and motivated Bill for the job, as he was elected to succeed me as director-sponsor in 1987. Bill then trained and motivated another BGSU person, Lynnette Porter, who then succeeded him in 1989.

Bill served in my presidential administration as assistant to the president for academic programs and we remained friends through our STC careers from then on.

Anyone who wishes to contribute to the fund that helps students attend and present papers at conferences can send their donation to BGSU Foundation, Mileti Alumni Center, Bowling Green, OH 43403; make sure to include the following on the memo line: "In Memory of Bill Coggin."

BILL LEAVITT *has written a variety of books on construction and architecture, history, and retirement planning. His latest book is entitled* Retirement: Life's Greatest Adventure. *The book contains guidance for how people thinking about retiring can best prepare for retirement and also suggestions for those who have retired but may not have discovered all the joys of retirement.*

Get the best of both worlds!
Publish natively to HTML5, EPUB3, KF8, MOBI, WebHelp, CHM and PDF
Author with best-in-class XML/DITA support

Upgrade to the power-packed Adobe FrameMaker 12

Now available at a monthly subscription of $29.99

Take advantage of Free Gold Support for 30-day trial

Try now

Request: Private demo | More information

Test drive now

Call 800-833-6687
(Monday-Friday, 5am-7pm PST)

Adobe, the Adobe logo, and FrameMaker are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.
© 2014 Adobe Systems Incorporated. All rights reserved.